program PASCALS(INPUT,OUTPUT,PRD,PRR);

{ author:N.Wirth, E.T.H. CH-8092 Zurich,1.3.76 }

{ modified by R.E.Berry

Department of computer studies

University of Lancaster

Variants of this program are used on

Data General Nova,Apple,and

Western Digital Microengine machines. }

{ further modified by M.Z.Jin

Department of Computer Science&Engineering BUAA,0ct.1989

}

{\* 定义常量 \*}

const nkw = 27; { no. of key words }

alng = 10; { no. of significant chars in identifiers }

llng = 121; { input line length }

emax = 322; { max exponent of real numbers }

emin = -292; { min exponent }

kmax = 15; { max no. of significant digits }

tmax = 100; { size of table }{\* 符号表范围 \*}

bmax = 20; { size of block-talbe }{\* 分程序表范围 \*}

amax = 30; { size of array-table }{\* 数组信息向量表范围 \*}

c2max = 20; { size of real constant table }{\* 实常数表范围 \*}

csmax = 30; { max no. of cases }

cmax = 800; { size of code }{\* Pcode表范围 \*}

lmax = 7; { maximum level }{\* Pcode指令x域范围 \*}

smax = 600; { size of string-table }{\* 字符串常量表范围 \*}

ermax = 58; { max error no. }{\* 错误信息表的范围 \*}

omax = 63; { highest order code }{\* Pcode指令编号域范围 \*}

xmax = 32767; { 2\*\*15-1 }{\* index范围 \*}

nmax = 32767; { 2\*\*15-1 }{\* Pcode指令y域范围 \*}

lineleng = 132; { output line length }

linelimit = 200;{\* 行数限制 \*}

stacksize = 1450;{\* 数据栈大小 \*}

{\* 定义各种基础数据类型 \*}

type symbol = ( intcon, realcon, charcon, stringcon,

notsy, plus, minus, times, idiv, rdiv, imod, andsy, orsy,

eql, neq, gtr, geq, lss, leq,

lparent, rparent, lbrack, rbrack, comma, semicolon, period,

colon, becomes, constsy, typesy, varsy, funcsy,

procsy, arraysy, recordsy, programsy, ident,

beginsy, ifsy, casesy, repeatsy, whilesy, forsy,

endsy, elsesy, untilsy, ofsy, dosy, tosy, downtosy, thensy);{\* 关键字的枚举，代表符号本身所属的类，为便于区分下文中称其为符号的“属性” \*}

index = -xmax..+xmax;{\* 子界，该类型的变量范围不会超过规定界限 \*}

alfa = packed array[1..alng]of char;{\* 长度为10的压缩数组，packed：元素存储空间不对齐 \*}

objecttyp = (konstant, vvariable, typel, prozedure, funktion ); {\* 所有种类的枚举 \*}

types = (notyp, ints, reals, bools, chars, arrays, records );{\* 所有类型的枚举 \*}

symset = set of symbol;

typset = set of types;

item = record

typ: types;

ref: index;

end;

{\* Pcode单个指令 \*}

order = packed record

f: -omax..+omax; {\* highest order code \*}

x: -lmax..+lmax; {\* maximum level \*}

y: -nmax..+nmax{\* 2\*\*15-1 \*}

end;

{\* 定义各种基础变量 \*}

var ch: char; { last character read from source program }

rnum: real; { real number from insymbol }{\* insymbol读到的实型 \*}

inum: integer; { integer from insymbol }{\* insymbol读到的整型 \*}

sleng: integer; { string length }

cc: integer; { character counter }

lc: integer; { program location counter }{\* Pcode的索引 \*}

ll: integer; { length of current line }{\* 用来标记当前行的行尾 \*}

errpos: integer; {\* 出错位置 \*}

t,a,b,sx,c1,c2:integer; { indices to tables }{\* 表格的栈顶指针 \*}

iflag, oflag, skipflag, stackdump, prtables: boolean;

sy: symbol; { last symbol read by insymbol }

errs: set of 0..ermax;{\* 记录错误编号的集合 \*}

id: alfa; { identifier from insymbol }{\* insymbol读到的符号名 \*}

progname: alfa;

stantyps: typset; {\* 标准类型的集合 \*}

constbegsys, typebegsys, blockbegsys, facbegsys, statbegsys: symset; {\* 表示各类代码开始的属性的集合 \*}

line: array[1..llng] of char; {\* lling : input line length121 \*}

key: array[1..nkw] of alfa; {\* 预定义的符号，共有27个 \*}

ksy: array[1..nkw] of symbol; {\* 预定义符号所对应的符号属性，表示这个符号本身 \*}

sps: array[char]of symbol; { special symbols }{\* 特殊符号 \*}

display: array[0..lmax] of integer; {\* display区，建立层次到btab的索引 \*}

tab: array[0..tmax] of { indentifier lable }{\* 符号表 \*}

packed record

name: alfa;

link: index;{\* 同一分程序中上一个标识符在tab表中的位置，用来查询 \*}

obj: objecttyp;{\* (konstant, vvariable, typel, prozedure, funktion ) \*}

typ: types;{\* (notyp, ints, reals, bools, chars, arrays, records )数据类型 \*}

ref: index;{\* 指向在atab或btab中登录的位置 \*}

normal: boolean;{\* 变量的区分：符号为变量形参时应填入false，为值形参或其他变量填入true \*}

lev: 0..lmax; {\* 层次 \*}

adr: integer {\* 具体含义在课本440页 \*}

end;

atab: array[1..amax] of { array-table }{\* 数组信息向量表 \*}

packed record

inxtyp,eltyp: types;

elref,low,high,elsize,size: index

end;

btab: array[1..bmax] of { block-table }{\* 分程序表 \*}

packed record

last, lastpar, psize, vsize: index

end;

stab: packed array[0..smax] of char; { string table }{\* 字符串常量表 \*}

rconst: array[1..c2max] of real;{\* 实常量表 \*}

code: array[0..cmax] of order;{\* Pcode表 \*}

psin,psout,prr,prd:text; { default in pascal p }{\* 用来输入输出的文件 \*}

inf, outf, fprr: string; {\* 文件名或文件路径 \*}

procedure errormsg; {\* 打印被编译的源程序中出错信息的摘要 \*}

var k : integer;

msg: array[0..ermax] of alfa; {\* 用来记录错误信息 \*}

begin

msg[0] := 'undef id '; msg[1] := 'multi def ';

msg[2] := 'identifier'; msg[3] := 'program ';

msg[4] := ') '; msg[5] := ': ';

msg[6] := 'syntax '; msg[7] := 'ident,var ';

msg[8] := 'of '; msg[9] := '( ';

msg[10] := 'id,array '; msg[11] := '( ';

msg[12] := '] '; msg[13] := '.. ';

msg[14] := '; '; msg[15] := 'func. type';

msg[16] := '= '; msg[17] := 'boolean ';

msg[18] := 'convar typ'; msg[19] := 'type ';

msg[20] := 'prog.param'; msg[21] := 'too big ';

msg[22] := '. '; msg[23] := 'type(case)';

msg[24] := 'character '; msg[25] := 'const id ';

msg[26] := 'index type'; msg[27] := 'indexbound';

msg[28] := 'no array '; msg[29] := 'type id ';

msg[30] := 'undef type'; msg[31] := 'no record ';

msg[32] := 'boole type'; msg[33] := 'arith type';

msg[34] := 'integer '; msg[35] := 'types ';

msg[36] := 'param type'; msg[37] := 'variab id ';

msg[38] := 'string '; msg[39] := 'no.of pars';

msg[40] := 'real numbr'; msg[41] := 'type ';

msg[42] := 'real type '; msg[43] := 'integer ';

msg[44] := 'var,const '; msg[45] := 'var,proc ';

msg[46] := 'types(:=) '; msg[47] := 'typ(case) ';

msg[48] := 'type '; msg[49] := 'store ovfl';

msg[50] := 'constant '; msg[51] := ':= ';

msg[52] := 'then '; msg[53] := 'until ';

msg[54] := 'do '; msg[55] := 'to downto ';

msg[56] := 'begin '; msg[57] := 'end ';

msg[58] := 'factor';

writeln(psout);{\* 打印空行 \*}

writeln(psout,'key words');

k := 0;

while errs <> [] do{\* 错误集合不是空集 \*}

begin

while not( k in errs )do k := k + 1; {\* 用k记录当前错误的编号 \*}

writeln(psout, k, ' ', msg[k] ); {\* 打印错误编号和错误信息 \*}

errs := errs - [k] {\* 错误已打印，从集合中删去 \*}

end {\* while errs \*}

end {\* errormsg \*} ;

procedure endskip; {\* 源程序出错后在被跳读的部分下面印出下划线标志 \*}

begin { underline skipped part of input }

while errpos < cc do {\* errpos：出错位置，cc：字符计数器character counter \*}

begin

write( psout, '-');

errpos := errpos + 1

end;

skipflag := false {\* 跳过代码并打印下划线之后，设置skipflag为false \*}

end {\* endskip \*};

procedure nextch; {\* read next character; process line end \*}{\* 读取下一字符，处理行结束符，印出被编译的源程序 \*}

begin

if cc = ll{\* 读到行尾 \*}

then begin

if eof( psin ){\* 如果已经读取完毕 \*}

then begin

writeln( psout );

writeln( psout, 'program incomplete' );

errormsg;{\* 打印错误信息 \*}

exit;

end;

if errpos <> 0 {\* 错误位置不为0说明有错误出现 \*}

then begin

if skipflag then endskip; {\* 如果跳过就加下划线 \*}

writeln( psout );

errpos := 0

end;

write( psout, lc: 5, ' ');{\* lc：Pcode的索引，输出代码位置;lc=5? \*}

ll := 0;{\* 长度指针置为0 \*}

cc := 0;{\* 字符指针置为0 \*}

while not eoln( psin ) do {\* 如果未读完，循环读取 \*}

begin

ll := ll + 1; {\* 当前行读取位置加一 \*}

read( psin, ch ); {\* 读入一个字符 \*}

write( psout, ch ); {\* 打印该字符 \*}

line[ll] := ch {\* 读到的字符写入line \*}

end;

ll := ll + 1;

readln( psin ); {\* 读空行 \*}

line[ll] := ' '; {\* 将line末尾设置为空格 \*}

writeln( psout );

end;

cc := cc + 1; {\* 字符计数器加一 \*}

ch := line[cc]; {\* 得到读到的字符 \*}

end {\* nextch \*};

procedure error( n: integer ); {\* 打印出错位置和出错编号 \*}

begin

if errpos = 0{\* 没有错误 \*}

then write ( psout, '\*\*\*\*' );

if cc > errpos

then begin

write( psout, ' ': cc-errpos, '^', n:2);{\* 格式化输出错误类型信息 \*}

errpos := cc + 3;

errs := errs +[n]{\* 当前错误加入错误列表 \*}

end

end {\* error \*};

procedure fatal( n: integer ); {\* 打印表格溢出信息，表格溢出则终止程序 \*}

var msg : array[1..7] of alfa;

begin

writeln( psout );

errormsg;{\* 打印错误信息摘要 \*}

msg[1] := 'identifier'; msg[2] := 'procedures';{\* 代表各种表格 \*}

msg[3] := 'reals '; msg[4] := 'arrays ';

msg[5] := 'levels '; msg[6] := 'code ';

msg[7] := 'strings ';

writeln( psout, 'compiler table for ', msg[n], ' is too small');

exit; {\* terminate compilation \*}

end {\* fatal \*};

procedure insymbol; {\* reads next symbol \*}{\* 读取下一单词符号，处理注释行 \*}

label 1,2,3;{\* 设置三个标签 \*}

var i,j,k,e: integer;{\* k表示位数，e表示指数 \*}

procedure readscale;{\* 处理实数的指数部分 \*}

var s,sign: integer;{\* s保存读到的数字 \*}

begin

nextch;{\* 读取下一个字符 \*}

sign := 1;{\* 符号标志 \*}

s := 0;{\* 初始化为0 \*}

if ch = '+'{\* 读到的字符为’+’ \*}

then nextch

else if ch = '-'{\* 读到的字符为’-’ ，则置符号标志为负 \*}

then begin

nextch;

sign := -1

end;

if not(( ch >= '0' )and (ch <= '9' )){\* 如果不是数字则报错（打印出错位置和错误编号） \*}

then error( 40 )

else repeat

s := 10\*s + ord( ord(ch)-ord('0'));{\* s保存读到的数字，ord是转换成ASCII码 \*}

nextch;

until not(( ch >= '0' ) and ( ch <= '9' ));

e := s\*sign + e{\* s\*sign是指数部分（字母’e’后面的）的指数，e是小数的指数，相加得到最终的e \*}

end { readscale };

procedure adjustscale;{\* 根据小数位数和指数大小求出实数数值 \*}

var s : integer;

d, t : real;

begin

if k + e > emax{\* 如果超过实数的指数部分上限则报错 \*}

then error(21)

else if k + e < emin{\* 如果低于实数的指数部分的下限则令实数为0 \*}

then rnum := 0

else begin

s := abs(e);{\* s保存指数部分绝对值，整个指数部分形式为t\*d^s，这里的e已经是小数部分指数与科学技术法指数的和 \*}

t := 1.0;

d := 10.0;

repeat

while not odd(s) do{\* odd函数判断是否为奇数，返回bool \*}

begin

s := s div 2;{\* 指数除以2 \*}

d := sqr(d){\* 与大多数语言不同，pascal中sqr()函数表示求平方 \*}

end;{\* 指数除以2，底数平方，结果等于原数值 \*}

s := s - 1;

t := d \* t{\* t乘一次底数d，底数d的指数减一，结果等于原数值 \*}

until s = 0;{\* 指数为0时，底数为计算结果 \*}

if e >= 0{\* 将指数部分与实数结合得到最终结果 \*}

then rnum := rnum \* t

else rnum := rnum / t

end

end { adjustscale };

procedure options;{\* 处理编译时的可选项 \*}

procedure switch( var b: boolean );{\* 处理编译可选项中的’+’’-’标志 \*}

begin

b := ch = '+';

if not b{\* 如果符号不是加号 \*}

then if not( ch = '-' ) {\* 也不是减号 \*}

then begin { print error message }{\* 当前if分之下有多条语句，因此要用begin和end括起来写成复合语句 \*}

while( ch <> '\*' ) and ( ch <> ',' ) do{\* 跳过除了’\*’和’,’之外的符号 \*}

nextch;

end

else nextch{\* ch = ‘-’ \*}

else nextch{\* b = true \*}

end { switch };

begin { options }{\* 定义在options层次下；该部分代码用来编译选项，t+表示打印各种表格并将prtables置为true，s+表示遇到运行出错时，会将stackdump置为true，并进行现场卸出打印，t-表示不打印表格并将prtables置为false，s-表示遇到运行出错时，会将stackdump置为false，且不进行现场卸出打印 \*}

repeat

nextch;

if ch <> '\*'{\* 编译选项写在注释里，因此一开始会有’\*’符号，需要跳过 \*}

then begin

if ch = 't'

then begin

nextch;

switch( prtables ){\* 若下一字符为’+’则置prtables为true，为’-’则置为false \*}

end

else if ch = 's'

then begin

nextch;

switch( stackdump ){\* 作用与上面类似 \*}

end;

end

until ch <> ','{\* 有’,’说明还有其他可选项，需要继续处理 \*}

end { options };

begin { insymbol }

1: while( ch = ' ' ) or ( ch = chr(9) ) do{\* 跳过空格和’\t’ \*}

nextch; { space & htab }

case ch of

'a','b','c','d','e','f','g','h','i',

'j','k','l','m','n','o','p','q','r',

's','t','u','v','w','x','y','z':

begin { identifier of wordsymbol }{\* 接case，如果是字母则开始识别标识符 \*}

k := 0;{\* 初始化位数 \*}

id := ' ';{\* 一共有10个空格，长度与alfa相等 \*}

repeat

if k < alng{\* 前十个字符读入id，遇到非字母或数字停止 \*}

then begin

k := k + 1;

id[k] := ch

end;

nextch

until not((( ch >= 'a' ) and ( ch <= 'z' )) or (( ch >= '0') and (ch <= '9' )));

i := 1;

j := nkw; { binary search }{\* nkw：no. of key words \*}

repeat

k := ( i + j ) div 2;

if id <= key[k]

then j := k - 1;

if id >= key[k]

then i := k + 1;

until i > j;

if i - 1 > j

then sy := ksy[k]{\* 获取当前id对应的sym（与key对应） \*}

else sy := ident{\* 若没有找到，说明不是保留字，是普通标识符；sy是枚举类型symbol的实例 \*}

end;

'0','1','2','3','4','5','6','7','8','9':

begin { number }{\* 接case，若以数字开头则视为数字 \*}

k := 0;

inum := 0;{\* inum是insymbol读取的数字 \*}

sy := intcon;{\* intcon表示整数 \*}

repeat

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

k := k + 1;

nextch

until not (( ch >= '0' ) and ( ch <= '9' ));

if( k > kmax ) or ( inum > nmax ){\* 如果位数超过上限或大小超过上限则报错并重置位数和大小 \*}

then begin

error(21);

inum := 0;

k := 0

end;

if ch = '.'{\* 读到小数点 \*}

then begin

nextch;

if ch = '.'{\* 如果读到连续两个’.’，说明读到的是数域而不是小数点 \*}

then ch := ':' {\* 连续两个’.’视为’:’ \*}

else begin{\* sy更新为实数 \*}

sy := realcon;

rnum := inum;

e := 0;{\* 指数 \*}

while ( ch >= '0' ) and ( ch <= '9' ) do

begin{\* 把小数表示为正整数和负指数 \*}

e := e - 1;

rnum := 10.0 \* rnum + (ord(ch) - ord('0'));

nextch

end;

if e = 0{\* 小数点后没有数字，报错 \*}

then error(40);

if ch = 'e'{\* 如果读到’e’，就开始处理指数部分 \*}

then readscale;

if e <> 0 then adjustscale{\* 计算结果 \*}

end

end

else if ch = 'e'

then begin

sy := realcon; {\* sy更新为实数 \*}

rnum := inum;

e := 0;

readscale;{\* 处理指数部分 \*}

if e <> 0

then adjustscale {\* 计算结果 \*}

end;

end;

':':

begin

nextch;

if ch = '='{\* 说明是’:=’赋值符号 \*}

then begin

sy := becomes;

nextch

end

else sy := colon{\* 说明是普通的’:’ \*}

end;

'<':

begin

nextch;

if ch = '='{\* 小于等于’<=’ \*}

then begin

sy := leq;

nextch

end

else

if ch = '>'{\* 不等于’<>’ \*}

then begin

sy := neq;

nextch

end

else sy := lss{\* 小于’<’ \*}

end;

'>':

begin

nextch;

if ch = '='{\* 大于等于’>=’ \*}

then begin

sy := geq;

nextch

end

else sy := gtr{\* 大于’>’ \*}

end;

'.':

begin

nextch;

if ch = '.'{\* 两个点’..’ \*}

then begin

sy := colon;{\* colon代表冒号’:’ \*}

nextch

end

else sy := period

end;

'''':

begin

k := 0;

2: nextch;

if ch = ''''

then begin

nextch;

if ch <> ''''

then goto 3{\* 字符串内容已经读完，跳转到3 \*}

end;

if sx + k = smax{\* sx(integer)：字符串常量表栈顶指针，smax：字符串表的大小字符串常量表的构造：stab:packed array[0..smax] of char；字符串表溢出则报错 \*}

then fatal(7);

stab[sx+k] := ch;

k := k + 1;

if cc = 1{\* cc=1表示nextch进入新行，说明当前行已读完 \*}

then begin {\* end of line \*}

k := 0;

end

else goto 2;{\* 当前行未读完，跳转到2继续读取 \*}

3: if k = 1{\* 引号内容长度为1 \*}

then begin

sy := charcon;{\* 代表字符 \*}

inum := ord( stab[sx] )

end

else if k = 0{\* 空引号（引号内显式输入了一个换行符），报错 \*}

then begin

error(38);

sy := charcon;{\* 代表字符 \*}

inum := 0

end

else begin

sy := stringcon;{\* 长度大于1，代表字符串 \*}

inum := sx;{\* 该字符串在stab的开始位置视为输入的整数 \*}

sleng := k;{\* 字符串长度设为位数 \*}

sx := sx + k

end

end;

'(':

begin

nextch;

if ch <> '\*'{\* 不是编译选项 \*}

then sy := lparent{\* 普通左括号 \*}

else begin {\* comment \*}

nextch;

if ch = '$'

then options;{\* 编译选项处理 \*}

repeat

while ch <> '\*' do nextch;

nextch

until ch = ')';

nextch;

goto 1{\* 编译选项处理完之后继续处理后面的字符 \*}

end

end;

'{\* ':

begin

nextch;

if ch = '$'

then options;{\* 编译选项处理 \*}

while ch <> ' \*}' do

nextch;

nextch;

goto 1{\* 编译选项处理完之后继续处理后面的字符 \*}

end;

'+', '-', '\*', '/', ')', '=', ',', '[', ']', ';':

begin

sy := sps[ch];{\* sps是特殊字符表，在procedure setup里初始化 \*}

nextch

end;

'$','"' ,'@', '?', '&', '^', '!':

begin

error(24);{\* 上述符号不能作为字符开头，报错 \*}

nextch;

goto 1{\* 跳转到1继续处理 \*}

end

end { case }

end { insymbol };

procedure enter(x0:alfa; x1:objecttyp; x2:types; x3:integer ); {\* 分程序外；将标准类型、过程、函数的名称登录到符号表（tab）中，其中x0为标识符，x1为标识符的种类（枚举），x2为标识符类型（枚举），x3为地址或大小（对于不同变量的含义不同） \*}

begin

t := t + 1; { enter standard identifier }{\* t是符号表的栈顶指针，初始化为-1 \*}

with tab[t] do{\* 一种简单的写法，避免了下面重复书写tab[t].name中的tab[t]. \*}

begin

name := x0;

link := t - 1;

obj := x1;

typ := x2;

ref := 0;

normal := true;

lev := 0;

adr := x3;

end

end; {\* enter \*}

procedure enterarray( tp: types; l,h: integer ); {\* 登录数组信息向量表 \*}

begin

if l > h{\* 上（h）下（l）界出错，报错 \*}

then error(27);

if( abs(l) > xmax ) or ( abs(h) > xmax ){\* 界限超过规定范围，报错；xmax：index的范围 \*}

then begin

error(27);

l := 0;

h := 0;

end;

if a = amax{\* 数组表已满，报错；amax：数组表的大小；a：数组表栈顶指针 \*}

then fatal(4)

else begin

a := a + 1;

with atab[a] do

begin

inxtyp := tp; {\* inxtyp：下标（索引）类型，可以是ints

、bools、chars \*}

low := l;{\* low：下界 \*}

high := h{\* high：上界 \*}

end

end

end { enterarray };

procedure enterblock; {\* 登录分程序表 \*}

begin

if b = bmax{\* 分程序表已满，报错；bmax：分程序表的大小；b：分程序表栈顶指针 \*}

then fatal(2)

else begin

b := b + 1;

btab[b].last := 0;{\* last：该分程序说明的当前（最后）一个标识符在tab表中的位置 \*}

btab[b].lastpar := 0;{\* lastpar：过程或函数的最后一个参数在tab表中的位置 \*}

end

end { enterblock };

procedure enterreal( x: real ); {\* 登录实常数表 \*}

begin

if c2 = c2max – 1 {\* 实常量表已满，报错；c2：实常量表栈顶指针 \*}

then fatal(3)

else begin

rconst[c2+1] := x;

c1 := 1;

while rconst[c1] <> x do

c1 := c1 + 1;

if c1 > c2

then c2 := c1{\* 以上操作保证实常量表中的常量（c2及之前的）不会重复 \*}

end

end { enterreal };

{\* 生成P代码指令 \*}

procedure emit( fct: integer );

begin

if lc = cmax{\* 程序进行到某行的定位（lc）超过Pcode表大小，报错 \*}

then fatal(6);

code[lc].f := fct; {\* code是order的数组，order有三个域：f: -omax..+omax（指令码范围）;x: -lmax..+lmax（层数范围）;y: -nmax..+nmax（数字大小范围） \*}

lc := lc + 1

end { emit };

procedure emit1( fct, b: integer ); {\* 生成P代码指令 \*}

begin

if lc = cmax{\* 程序进行到某行的定位（lc）超过Pcode表大小，报错 \*}

then fatal(6);

with code[lc] do

begin

f := fct;

y := b;

end;

lc := lc + 1

end { emit1 };

procedure emit2( fct, a, b: integer );

begin

if lc = cmax then fatal(6); {\* 程序进行到某行的定位（lc）超过Pcode表大小，报错 \*}

with code[lc] do

begin

f := fct;

x := a;

y := b

end;

lc := lc + 1;

end { emit2 };

procedure printtables; {\* 打印编译生成的符号表、分程序表、实常数表、Pcode \*}

var i: integer;

o: order;

mne: array[0..omax] of

packed array[1..5] of char;{\* 存储指令符号 \*}

begin

mne[0] := 'LDA '; mne[1] := 'LOD '; mne[2] := 'LDI ';

mne[3] := 'DIS '; mne[8] := 'FCT '; mne[9] := 'INT ';

mne[10] := 'JMP '; mne[11] := 'JPC '; mne[12] := 'SWT ';

mne[13] := 'CAS '; mne[14] := 'F1U '; mne[15] := 'F2U ';

mne[16] := 'F1D '; mne[17] := 'F2D '; mne[18] := 'MKS ';

mne[19] := 'CAL '; mne[20] := 'IDX '; mne[21] := 'IXX ';

mne[22] := 'LDB '; mne[23] := 'CPB '; mne[24] := 'LDC ';

mne[25] := 'LDR '; mne[26] := 'FLT '; mne[27] := 'RED ';

mne[28] := 'WRS '; mne[29] := 'WRW '; mne[30] := 'WRU ';

mne[31] := 'HLT '; mne[32] := 'EXP '; mne[33] := 'EXF ';

mne[34] := 'LDT '; mne[35] := 'NOT '; mne[36] := 'MUS ';

mne[37] := 'WRR '; mne[38] := 'STO '; mne[39] := 'EQR ';

mne[40] := 'NER '; mne[41] := 'LSR '; mne[42] := 'LER ';

mne[43] := 'GTR '; mne[44] := 'GER '; mne[45] := 'EQL ';

mne[46] := 'NEQ '; mne[47] := 'LSS '; mne[48] := 'LEQ ';

mne[49] := 'GRT '; mne[50] := 'GEQ '; mne[51] := 'ORR ';

mne[52] := 'ADD '; mne[53] := 'SUB '; mne[54] := 'ADR ';

mne[55] := 'SUR '; mne[56] := 'AND '; mne[57] := 'MUL ';

mne[58] := 'DIV '; mne[59] := 'MOD '; mne[60] := 'MUR ';

mne[61] := 'DIR '; mne[62] := 'RDL '; mne[63] := 'WRL ';

writeln(psout);

writeln(psout);

writeln(psout);

writeln(psout,' identifiers link obj typ ref nrm lev adr');{\* 打印符号表 \*}

writeln(psout);

for i := btab[1].last to t do{\* btab[1].last初始化为28，即非预定义符号开始的地方；tab表的0和28位置是空符号（没有名称） \*}

with tab[i] do

writeln( psout, i,' ', name, link:5, ord(obj):5, ord(typ):5,ref:5, ord(normal):5,lev:5,adr:5);

writeln( psout );

writeln( psout );

writeln( psout );

writeln( psout, 'blocks last lpar psze vsze' );{\* 打印分程序表 \*}

writeln( psout );

for i := 1 to b do

with btab[i] do

writeln( psout, i:4, last:9, lastpar:5, psize:5, vsize:5 );

writeln( psout );

writeln( psout );

writeln( psout );

writeln( psout, 'arrays xtyp etyp eref low high elsz size');{\* 打印数组信息向量表 \*}

writeln( psout );

for i := 1 to a do

with atab[i] do

writeln( psout, i:4, ord(inxtyp):9, ord(eltyp):5, elref:5, low:5, high:5, elsize:5, size:5);

writeln( psout );

writeln( psout );

writeln( psout );

writeln( psout, 'code:');{\* 打印Pcode表 \*}

writeln( psout );

for i := 0 to lc-1 do

begin

write( psout, i:5 );

o := code[i];

write( psout, mne[o.f]:8, o.f:5 );{\* 打印助记符（LDA）和操作码（0） \*}

if o.f < 31

then if o.f < 4

then write( psout, o.x:5, o.y:5 ){\* 打印操作数 \*}

else write( psout, o.y:10 )

else write( psout, ' ' );

writeln( psout, ',' ){\* 行尾打印’,’ \*}

end;

writeln( psout );

writeln( psout, 'Starting address is ', tab[btab[1].last].adr:5 )

end { printtables };

procedure block( fsys: symset; isfun: boolean; level: integer );{\* 分析处理分程序；symset：symbol的集合；level为当前层号 \*}

type conrec = record{\* 这样的结构体可以根据不同的type来保存不同的其他域 \*}

case tp: types of

ints, chars, bools : ( i:integer );

reals :( r:real )

end;

var dx : integer ; {\* data allocation index \*}{\* 数据分配的索引，初值为5，每个分程序在运行栈S中的数据区开头应留出5个单元，用来存放函数结果、返回地址（RA）、静态链（SL）、动态连（DL）以及该过程或函数在符号表中登记的位置 \*}

prt: integer ; { t-index of this procedure }

prb: integer ; { b-index of this procedure }

x : integer ;

procedure skip( fsys:symset; n:integer);{\* 跳读源程序，直到取来的符号属于给出的符号集为止，并打印出错标志 \*}

begin

error(n);{\* 打印出错的位置以及错误编号 \*}

skipflag := true;

while not ( sy in fsys ) do

insymbol;

if skipflag then endskip{\* endskip：出错后在跳读的部分下面印出下划线标志 \*}

end { skip };

procedure test( s1,s2: symset; n:integer );{\* 测试当前符号是否合法，若不合法，打印出错标志并进行跳读 \*}

begin

if not( sy in s1 )

then skip( s1 + s2, n )

end { test };

procedure testsemicolon;{\* 测试当前符号是否为’;’ \*}

begin

if sy = semicolon{\* 如果是继续读取 \*}

then insymbol

else begin{\* 如果不是打印出错位置和信息 \*}

error(14);

if sy in [comma, colon]{\* 如果是’,’或’:’就继续读取 \*}

then insymbol

end;

test( [ident] + blockbegsys, fsys, 6 ){\* ident：普通标识符；blockbegsys是标志分程序开始的symbol的集合 \*}

end { testsemicolon };

procedure enter( id: alfa; k:objecttyp );{\* 分程序内；在符号表中登录分程序说明部分出现的名字 \*}

var j,l : integer;

begin

if t = tmax{\* 符号表满，报错 \*}

then fatal(1)

else begin

tab[0].name := id;{\* 给 tab[0]赋值是为了即使没有找到，也会让寻找过程在tab[0]处停止（函数和过程名的link值为0，也就是说根据link回溯到tab里的函数和过程名时一定会回到tab[0]） \*}

j := btab[display[level]].last;{\* display：btab的索引表；获取当前层最后一个标识符在tab表中的位置 \*}

l := j;

while tab[j].name <> id do{\* 若与当前id不相等，则继续获取上一个标识符直到相等 \*}

j := tab[j].link;

if j <> 0{\* 说明当前符号已经在当前分程序定义过又重定义，打印错误位置和信息 \*}

then error(1)

else begin{\* 若没有重定义就加入符号表 \*}

t := t + 1;

with tab[t] do

begin

name := id;

link := l;{\* 原本当前层的最后一个标识符的位置成为新的标识符上一个标识符（link）的位置 \*}

obj := k;

typ := notyp;

ref := 0;

lev := level;

adr := 0;

normal := false { initial value }{\* 初始化为false \*}

end;

btab[display[level]].last := t{\* 更新指向当前层最后一个标识符的指针的位置 \*}

end

end

end { enter };

function loc( id: alfa ):integer;{\* 查找标识符在符号表中的位置 \*}

var i,j : integer; { locate if in table }

begin

i := level;

tab[0].name := id; { sentinel }

repeat

j := btab[display[i]].last;{\* 获取最后一个符号位置 \*}

while tab[j].name <> id do{\* 逐层寻找 \*}

j := tab[j].link;

i := i - 1;

until ( i < 0 ) or ( j <> 0 );{\* 当所有层都找完或者找到为止 \*}

if j = 0{\* 所有层都找过但是没找到，打印错误位置和信息 \*}

then error(0);

loc := j{\* 若找到，返回id在符号表中的位置 \*}

end { loc } ;

procedure entervariable;{\* 将变量名登录到符号表中 \*}

begin

if sy = ident{\* 如果是标识符就登录到符号表中 \*}

then begin

enter( id, vvariable );{\* 种类是变量 \*}

insymbol

end

else error(2){\* 如果不是变量，报错 \*}

end { entervariable };

procedure constant( fsys: symset; var c: conrec );{\* 处理程序中出现的常量，并由参数c返回该常量的类型和数值 \*}

var x, sign : integer;

begin

c.tp := notyp;{\* 初始化c \*}

c.i := 0;

test( constbegsys, fsys, 50 );{\* 测试当前标识符是否合法 \*}

if sy in constbegsys{\* constbegsys：表示常量声明开始的符号 \*}

then begin

if sy = charcon{\* 字符常量 \*}

then begin

c.tp := chars;

c.i := inum;{\* 字符常量的inum值为其ASCII码 \*}

insymbol

end

else begin

sign := 1;

if sy in [plus, minus]{\* ‘+’或者’-’ \*}

then begin

if sy = minus{\* ‘-’则改变sign \*}

then sign := -1;

insymbol

end;

if sy = ident{\* 普通标识符 \*}

then begin

x := loc(id);{\* 获得其在符号表中的位置 \*}

if x <> 0{\* 符号表中记录了该标识符 \*}

then

if tab[x].obj <> constant {\* 该标识符不是常量，打印出错位置和信息 \*}

then error(25)

else begin{\* 若是常量，给c的域赋值 \*}

c.tp := tab[x].typ;

if c.tp = reals

then c.r := sign\*rconst[tab[x].adr]{\* 符号表中实常量名的adr域不是它的实数值而是指向该实数在实常量表位置的指针 \*}

else c.i := sign\*tab[x].adr

end;

insymbol

end

else if sy = intcon{\* 整数常量，给c的域赋值 \*}

then begin

c.tp := ints;

c.i := sign\*inum;

insymbol

end

else if sy = realcon{\* 实常量，给c的域赋值 \*}

then begin

c.tp := reals;

c.r := sign\*rnum;

insymbol

end

else skip(fsys,50){\* 若不是上述类型，直接跳过，50是错误编号 \*}

end;{\* else begin（不是字符常量） \*}

test(fsys,[],6){\* 继续测试下一行是否合法 \*}

end{ if sy in constbegsys }

end { constant };

procedure typ( fsys: symset; var tp: types; var rf,sz:integer );{\* 处理类型描述，由参数得到它的类型（tp），指向类型详细信息表的指针（rf）和该类型大小（sz） \*}

var eltp : types;

elrf, x : integer;

elsz, offset, t0, t1 : integer;

procedure arraytyp( var aref, arsz: integer );{\* 处理数组类型，由参数返回指向该数组信息向量表的指针（aref）和数组大小（arsz） \*}

var eltp : types;{\* 数组的元素类型 \*}

low, high : conrec;{\* 数组的上下界 \*}

elrf, elsz: integer;{\* 指向数组信息向量表的指针和数组大小 \*}

begin

constant( [colon, rbrack, rparent, ofsy] + fsys, low );{\* 获得数组编号的下界记录在low中 \*}

if low.tp = reals{\* 如果数组下界为实数则打印错误位置和信息，把数组转换成整型数组并设置下界为0 \*}

then begin

error(27);

low.tp := ints;

low.i := 0

end;

if sy = colon{\* 如果下界后面是’:’，即’..’ ，则继续读取数组上界 \*}

then insymbol

else error(13);{\* 如果不是’:’，打印错误位置和信息 \*}

constant( [rbrack, comma, rparent, ofsy ] + fsys, high );{\* 获得数组编号上界记录在high中 \*}

if high.tp <> low.tp{\* 上界类型不是整数，则打印错误位置和信息，并设置上界等于下界 \*}

then begin

error(27);

high.i := low.i

end;

enterarray( low.tp, low.i, high.i );{\* 把数组信息登录在数组信息向量表atab中 \*}

aref := a;{\* 获取当前数组在atab中的位置 \*}

if sy = comma{\* 如果读取到’,’，说明该数组为多维数组 \*}

then begin

insymbol;

eltp := arrays;{\* 当前数组的元素类型也是数组 \*}

arraytyp( elrf, elsz ){\* 递归调用，处理数组信息 \*}

end

else begin{\* 如果没有’,’，说明仅是一维数组 \*}

if sy = rbrack{\* 读到’]’，说明数组声明结束 \*}

then insymbol

else begin{\* 若不是’]’，打印错误位置和信息 \*}

error(12);

if sy = rparent

then insymbol

end;

if sy = ofsy{\* 如果读到了’of’ \*}

then insymbol

else error(8);{\* 若没有读到下一个字符，打印错误位置和信息 \*}

typ( fsys, eltp, elrf, elsz ){\* 处理当前符号类型 \*}

end;

with atab[aref] do{\* 把当前数组信息记录在数组信息向量表atab，aref是当前数组位置 \*}

begin

arsz := (high-low+1) \* elsz;

size := arsz;

eltyp := eltp;

elref := elrf;

elsize := elsz

end

end { arraytyp };

begin { typ }

tp := notyp;{\* 初始化变量类型 \*}

rf := 0; {\* 初始化在符号表的指针 \*}

sz := 0; {\* 初始化变量的大小 \*}

test( typebegsys, fsys, 10 );{\* typebegsys：类型声明开始的符号；判断是否是类型声明开始的符号，不是则报错 \*}

if sy in typebegsys

then begin

if sy = ident{\* 如果是普通标识符 \*}

then begin

x := loc(id);{\* 查找该标识符在符号表中的位置 \*}

if x <> 0{\* 找到该标识符 \*}

then with tab[x] do

if obj <> typel{\* 类型不是typel，打印错误位置和信息 \*}

then error(29)

else begin{\* 如果是，获得标识符信息 \*}

tp := typ;

rf := ref;{\* ref：该标识符的位置 \*}

sz := adr;{\* 这里的adr：在符号表中的相对地址 \*}

if tp = notyp{\* 如果未定义标识符的类型，则打印错误位置和信息 \*}

then error(30)

end;

insymbol

end

else if sy = arraysy{\* 当前符号是数组 \*}

then begin

insymbol;

if sy = lbrack{\* 读到’[’则继续读 \*}

then insymbol

else begin{\* 若不是则打印错误位置和信息 \*}

error(11);

if sy = lparent{\* 若读到’(’ ，容错，可以接受，继续读取 \*}

then insymbol

end;

tp := arrays;{\* 当前符号类型设置为数组 \*}

arraytyp(rf,sz){\* 保存数组，获取数组在atab中的位置和数组大小 \*}

end

else begin {\* records \*}{\* 还剩一种可能就是recordsy \*}

insymbol;

enterblock;{\* 登录分程序表 \*}

tp := records;{\* 当前符号类型设置为记录 \*}

rf := b;{\* 用rf记录当前记录在分程序表中的位置 \*}

if level = lmax{\* 当前层已经是最大层（block有嵌套层数限制），发生层溢出，报错 \*}

then fatal(5);

level := level + 1;

display[level] := b;{\* 建立最大层到分程序表的索引 \*}

offset := 0;

while not ( sy in fsys - [semicolon,comma,ident]+ [endsy] ) do{\* 声明record类型内的变量 \*}

begin { field section }

if sy = ident{\* 普通标识符 \*}

then begin

t0 := t;

entervariable; {\* 将变量登录符号表 \*}

while sy = comma do{\* 若遇到’,’则继续读入并登录变量 \*}

begin

insymbol;

entervariable

end;

if sy = colon{\* 若遇到’:’，继续读取变量类型 \*}

then insymbol

else error(5);{\* 如果没读到’,’或’:’，打印错误位置和信息 \*}

t1 := t;{\* 记录全部读完之后符号表栈顶位置，t在entervariable中更新 \*}

typ( fsys + [semicolon, endsy, comma,ident], eltp, elrf, elsz );{\* 递归调用typ来处理这些读到的变量 \*}

while t0 < t1 do{\* 填写符号表从t0到t1缺少的部分 \*}

begin

t0 := t0 + 1;

with tab[t0] do{\* 记录该变量信息 \*}

begin

typ := eltp;

ref := elrf;

normal := true;

adr := offset;{\* offset：偏移量 \*}

offset := offset + elsz

end

end

end; {\* sy = ident \*}

if sy <> endsy{\* 读到’end’，说明记录的成员变量声明结束 \*}

then begin

if sy = semicolon{\* 记录声明末尾应该读到’;’ \*}

then insymbol

else begin{\* 没有读到’;’，打印错误位置和信息 \*}

error(14);

if sy = comma{\* 读到’,’，容错，可以接受，继续读取 \*}

then insymbol

end;

test( [ident,endsy, semicolon],fsys,6 ){\* 测试读到的符号是否合法 \*}

end

end; { field section }{\* 记录的成员变量处理完成 \*}

btab[rf].vsize := offset;{\* 记录局部变量、参数和display区在分程序表中占用的总空间 \*}

sz := offset;{\* 同上 \*}

btab[rf].psize := 0;{\* 记录的参数所占空间的大小，为0 \*}

insymbol;

level := level - 1

end; { record }

test( fsys, [],6 ){\* 测试读到的符号是否合法 \*}

end;

end { typ };

procedure parameterlist; { formal parameter list }{\* 处理过程或函数说明中的形参表，将形参及其有关信息登录到符号表中 \*}

var tp : types;{\* 形参的类型 \*}

valpar : boolean;{\* 值形参或变量形参 \*}

rf, sz, x, t0 : integer; {\* rf：符号表位置；sz：大小 \*}

begin

insymbol;

tp := notyp;{\* 初始化 \*}

rf := 0;

sz := 0;

test( [ident, varsy], fsys+[rparent], 7 );{\* 测试当前符号是否合法 \*}

while sy in [ident, varsy] do{\* 如果读到的是普通标识符或者’var’ \*}

begin

if sy <> varsy{\* 如果是普通标识符 \*}

then valpar := true {\* 普通标识符表明是值形参，设置为true \*}

else begin{\* 如果读到’var’，继续读参数名 \*}

insymbol;

valpar := false {\* 变量形参，设置为false \*}

end;

t0 := t;{\* 记录符号表栈顶位置 \*}

entervariable;{\* 参数登录到符号表 \*}

while sy = comma do{\* 如果读到’,’，则继续读 \*}

begin

insymbol;

entervariable; {\* 参数登录到符号表 \*}

end;

if sy = colon{\* 如果读到’:’ \*}

then begin

insymbol;

if sy <> ident{\* 如果不是普通标识符，打印错误位置和信息 \*}

then error(2)

else begin

x := loc(id);{\* 是普通标识符，获取其在符号表中的位置 \*}

insymbol;

if x <> 0{\* 在符号表中找到 \*}

then with tab[x] do

if obj <> typel{\* 如果不是typel，则打印错误位置和信息 \*}

then error(29)

else begin{\* 如果是，获取其类型和在符号表中的位置 \*}

tp := typ;

rf := ref;

if valpar{\* 如果是值形参，sz保存其分配的存储单元的相对地址 \*}

then sz := adr

else sz := 1{\* 如果是变量形参，sz设置为1 \*}

end;

end;

test( [semicolon, rparent], [comma,ident]+fsys, 14 ){\* 测试当前符号是否合法 \*}

end

else error(5);{\* 如果不是冒号，打印错误位置和信息 \*}

while t0 < t do{\* 符号表中从t0到t的元素还没有赋值，为其赋值 \*}

begin

t0 := t0 + 1;

with tab[t0] do

begin

typ := tp;

ref := rf;

adr := dx{\* dx：变量存储分配的索引（相对位置）；adr对于变量（包括形参）名，应填入该变量（或形参）在运行栈S中分配存储单元的相对地址 \*};

lev := level;{\* 定义所在的层次，和运行栈S中的调用层次不同；主程序main的层次为1 \*}

normal := valpar;{\* 标志其是值形参还是变量形参 \*}

dx := dx + sz{\* 更新存储分配的指针，若为值形参则加上相对地址，若为变量形参则加上1 \*}

end

end;

if sy <> rparent{\* 不是’)’ \*}

then begin

if sy = semicolon {\* 如果是’;’ \*}

then insymbol

else begin{\* 如果不是则打印错误位置和信息 \*}

error(14);

if sy = comma{\* 如果是’,’，容错，继续读下一个符号 \*}

then insymbol

end;

test( [ident, varsy],[rparent]+fsys,6){\* 测试当前符号是否为普通标识符或者’var’ \*}

end

end { while };

if sy = rparent{\* 如果是’)’ \*}

then begin

insymbol;

test( [semicolon, colon],fsys,6 ){\* 测试当前符号是否为分号或冒号 \*}

end

else error(4){\* 如果不是’)’，打印错误位置和信息 \*}

end { parameterlist };

procedure constdec;{\* 处理常量定义，将常量名及其相应信息填入符号表 \*}

var c : conrec;

begin

insymbol;

test([ident], blockbegsys, 2 );{\* 测试当前符号是不是普通标识符 \*}

while sy = ident do

begin

enter(id, konstant);{\* 登录到符号表中，种类为常量 \*}

insymbol;

if sy = eql{\* 如果是’=’ \*}

then insymbol

else begin{\* 如果不是，打印错误位置和信息 \*}

error(16);

if sy = becomes{\* 如果是’:=’，容错，继续读 \*}

then insymbol

end;

constant([semicolon,comma,ident]+fsys,c);{\* 获得常量的类型和数值 \*}

tab[t].typ := c.tp;{\* 把类型填入符号表 \*}

tab[t].ref := 0;{\* 常量的ref为0 \*}

if c.tp = reals{\* 如果类型为实数 \*}

then begin

enterreal(c.r);{\* 把实数登录到实常数表 \*}

tab[t].adr := c1;{\* 实常量的adr保存其在实常数表的位置 \*}

end

else tab[t].adr := c.i;{\* 如果是整数，直接保存它的值（i域中保存了c的数值） \*}

testsemicolon{\* 检测当前符号是不是’;’ \*}

end

end {\* constdec \*};

procedure typedeclaration;{\* 处理类型定义，将类型名及其相应信息填入符号表 \*}

var tp: types;

rf, sz, t1 : integer;

begin

insymbol;

test([ident], blockbegsys,2 );{\* 检测当前符号是不是普通标识符 \*}

while sy = ident do{\* 如果是普通标识符 \*}

begin

enter(id, typel);{\* 登录到符号表中，种类为type \*}

t1 := t;{\* 记录符号表栈顶指针 \*}

insymbol;

if sy = eql{\* 如果是’=’ \*}

then insymbol

else begin{\* 如果不是，打印错误位置和信息 \*}

error(16);

if sy = becomes{\* 如果读到’:=’，容错，继续读 \*}

then insymbol

end;

typ( [semicolon,comma,ident]+fsys, tp,rf,sz );{\* 处理该变量、获得指向类型详细信息表的指针、在符号表的位置、大小 \*}

with tab[t1] do{\* 给表中对应元素赋值 \*}

begin

typ := tp;

ref := rf;

adr := sz

end;

testsemicolon{\* 检测是否为分号 \*}

end

end { typedeclaration };

procedure variabledeclaration;{\* 处理变量定义，将变量名及其相应信息填入符号表 \*}

var tp : types;

t0, t1, rf, sz : integer;

begin

insymbol;

while sy = ident do{\* 如果是普通标识符 \*}

begin

t0 := t;{\* 获得符号表栈顶指针 \*}

entervariable;{\* 将变量登录符号表 \*}

while sy = comma do{\* 若读到’,’ \*}

begin

insymbol;{\* 继续读变量并登录到符号表中 \*}

entervariable;

end;

if sy = colon{\* 如果读到’:’ \*}

then insymbol

else error(5);{\* 如果不是，打印错误位置及信息 \*}

t1 := t;

typ([semicolon,comma,ident]+fsys, tp,rf,sz );{\* 获取当前变量的类型、指向类型详细信息表的指针、大小 \*}

while t0 < t1 do{\* 符号表t0到t1之间的元素填表 \*}

begin

t0 := t0 + 1;

with tab[t0] do

begin

typ := tp;

ref := rf;

lev := level;

adr := dx;{\* 变量存储分配的索引 \*}

normal := true;

dx := dx + sz{\* 更新存储分配的指针位置 \*}

end

end;

testsemicolon{\* 检测是否为分号 \*}

end

end { variabledeclaration };

procedure procdeclaration;{\* 处理过程或函数说明，将过程（或函数）名填入符号表中，递归调用block分析处理程序（层次level+1） \*}

var isfun : boolean;

begin

isfun := sy = funcsy;{\* 当前符号是不是’function’ \*}

insymbol;

if sy <> ident{\* 读到的符号不是普通标识符，打印错误位置及信息 \*}

then begin

error(2);

id :=' '

end;

if isfun{\* 如果是’function’ \*}

then enter(id,funktion){\* 登录到符号表中，种类为function \*}

else enter(id,prozedure);{\* 登录到符号表中，种类为procedure \*}

tab[t].normal := true; {\* 值形参或其他变量应填入true \*}

insymbol;

block([semicolon]+fsys, isfun, level+1 );{\* 递归处理，层次加一 \*}

if sy = semicolon{\* 如果读到’;’ \*}

then insymbol

else error(14);{\* 否则打印错误位置及信息 \*}

emit(32+ord(isfun)) {\* exit \*}{\* 生成Pcode指令，退出函数或过程 \*}

end { proceduredeclaration };

procedure statement( fsys:symset );{\* 分析处理各种语句 \*}

var i : integer;

procedure expression(fsys:symset; var x:item); forward;{\* 分析处理表达式，由参数（x）返回求值结果的类型；forward是允许向前调用的关键字，相当于提前声明一个同级的过程 \*}

procedure selector(fsys:symset; var v:item);{\* 处理结构变量：数组下标变量或记录成员变量；item是record类型，有两个域：typ:types和ref:index \*}

var x : item;

a,j : integer;

begin { sy in [lparent, lbrack, period] }{\* 当前符号应为’(’、’[’或者’.’ \*}

repeat

if sy = period{\* 如果当前符号是’.’，由于记录的引用方式是记录名.域名，因此当前读到的应该是记录 \*}

then begin

insymbol; { field selector }

if sy <> ident{\* 当前符号不是普通标识符，打印错误位置及信息 \*}

then error(2)

else begin

if v.typ <> records{\* 如果当前处理的不是记录，打印错误位置及信息 \*}

then error(31)

else begin {\* search field identifier \*}

j := btab[v.ref].last;{\* 获得记录在分程序表中最后一个标识符在符号表中的位置 \*}

tab[0].name := id;

while tab[j].name <> id do{\* 通过link在符号表中向上寻找 \*}

j := tab[j].link;

if j = 0{\* 没有找到，当前标识符不在符号表中，打印错误位置及信息 \*}

then error(0);

v.typ := tab[j].typ{\* 如果找到，记录下其类型和其在相应类型详细信息表的指针 \*};

v.ref := tab[j].ref;

a := tab[j].adr;{\* 记录域名的adr是它相对该记录变量起始地址的位移量 \*}

if a <> 0{\* 如果位移不是0，就产生一条Pcode指令来计算位移 \*}

then emit1(9,a)

end;

insymbol

end

end

else begin {\* array selector \*}{\* 读到数组 \*}

if sy <> lbrack{\* 如果不是’[’，打印错误位置及信息 \*}

then error(11);

repeat

insymbol;{\* 继续读，多维数组 \*}

expression( fsys+[comma,rbrack],x);{\* 递归调用，处理数组，结果保存在x中 \*}

if v.typ <> arrays{\* 如果传入的类型不是数组，打印错误位置及信息 \*}

then error(28)

else begin

a := v.ref;{\* 获得当前数组在数组信息向量表的位置 \*}

if atab[a].inxtyp <> x.typ{\* 传入的下标的类型与数组的下标类型不同，打印错误位置及信息 \*}

then error(26)

else if atab[a].elsize = 1{\* 元素大小为1，说明是变量形参 \*}

then emit1(20,a){\* 若是变量形参，生成Pcode指令找到其地址 \*}

else emit1(21,a);{\* 若是值形参，也生成Pcode指令找到其地址 \*}

v.typ := atab[a].eltyp;{\* 获得数组元素的类型和数组元素在atab的位置 \*}

v.ref := atab[a].elref

end

until sy <> comma;{\* 如果读到的不是’,’，说明数组不再有更高维度 \*}

if sy = rbrack{\* 如果是’]’ \*}

then insymbol

else begin{\* 如果不是，打印错误位置及信息 \*}

error(12);

if sy = rparent{\* 如果是’)’，容错，继续读 \*}

then insymbol

end

end

until not( sy in[lbrack, lparent, period]);{\* 一直循环读取直到所有的数组和记录都找到地址 \*}

test( fsys,[],6){\* 测试当前符号是否合法 \*}

end { selector };

procedure call( fsys: symset; i:integer );{\* 处理非标准的过程或函数调用，i为需要调用的过程或者函数名在符号表中的位置 \*}

var x : item;

lastp,cp,k : integer;

begin

emit1(18,i); { mark stack }

lastp := btab[tab[i].ref].lastpar{\* 保存当前过程或函数的最后一个参数在符号表中的位置 \*};

cp := i;{\* 记录该过程或函数在符号表中的位置 \*}

if sy = lparent {\* 如果读到’(’ \*}

then begin { actual parameter list }

repeat

insymbol;

if cp >= lastp{\* 当前符号的位置大于最后一个参数的位置，打印错误位置及信息 \*}

then error(39)

else begin

cp := cp + 1;

if tab[cp].normal{\* 如果传入的是值形参或者其他参数 \*}

then begin { value parameter }

expression( fsys+[comma, colon,rparent],x);{\* 递归调用 \*}

if x.typ = tab[cp].typ{\* 得到的参数的类型和符号表元素的类型相同 \*}

then begin

if x.ref <> tab[cp].ref{\* 参数指向的btab的位置和符号表记录的btab的位置不同，打印错误位置及信息 \*}

then error(36)

else if x.typ = arrays{\* 如果参数的类型是数组 \*}

then emit1(22,atab[x.ref].size){\* 生成装入指令，将实参表达式的值或地址放到预留的参数位置 \*}

else if x.typ = records{\* 如果参数的类型是记录 \*}

then emit1(22,btab[x.ref].vsize){\* 生成装入指令，将实参表达式的值或地址放到预留的参数位置 \*}

end

else if ( x.typ = ints ) and ( tab[cp].typ = reals ){\* 如果参数的类型是整型但是符号表中记录的是实型 \*}

then emit1(26,0){\* 生成类型转化指令，整数转化为浮点数 \*}

else if x.typ <> notyp{\* 参数的类型不能获取，打印错误位置及信息 \*}

then error(36);

end

else begin {\* variable parameter \*}{\* 如果是变量形参 \*}

if sy <> ident{\* 变量形参必为标识符 \*}

then error(2)

else begin

k := loc(id);{\* 记录当前变量在符号表的位置 \*}

insymbol;

if k <> 0{\* 如果不是0，说明id在符号表中 \*}

then begin

if tab[k].obj <> vvariable{\* 如果符号表中记录的当前变量的种类不是变量，打印错误位置及信息 \*}

then error(37);

x.typ := tab[k].typ;{\* 获取符号表记录的当前变量的类型和位置 \*}

x.ref := tab[k].ref;

if tab[k].normal{\* 如果是值形参 \*}

then emit2(0,tab[k].lev,tab[k].adr){\* 将地址传入栈顶 \*}

else emit2(1,tab[k].lev,tab[k].adr);{\* 如果是变量形参，将其值传入栈顶，传入的是地址还是值用Pcode指令编号来区分（0和1） \*}

if sy in [lbrack, lparent, period]{\* 如果是’[’、’(’或者’.’ \*}

then selector(fsys+[comma,colon,rparent],x);{\* 处理数组或记录 \*}

if ( x.typ <> tab[cp].typ ) or ( x.ref <> tab[cp].ref ){\* 如果参数的类型和地址与符号表中记录的不同，打印错误位置及信息 \*}

then error(36)

end

end

end {\* variable parameter \*}

end;

test( [comma, rparent],fsys,6){\* 测试当前sym是否合法 \*}

until sy <> comma;{\* 一直循环直到不出现’,’，停止循环 \*}

if sy = rparent{\* 如果读到’)’，继续读取，否则打印错误位置及信息 \*}

then insymbol

else error(4)

end;

if cp < lastp{\* 如果当前符号位置没有达到当前过程或函数的最后一个参数在符号表中的位置，打印错误位置及信息 \*}

then error(39); { too few actual parameters }

emit1(19,btab[tab[i].ref].psize-1 );{\* 生成Pcode指令，调用过程或函数 \*}

if tab[i].lev < level{\* 如果调用函数/过程的层次小于当前层 \*}

then emit2(3,tab[i].lev, level ){\* 生成Pcode指令，更新display，改变调用函数/过程的层次 \*}

end { call };

function resulttype( a, b : types) :types;{\* 处理整型或实型两个操作数运算的类型转换 \*}

begin

if ( a > reals ) or ( b > reals ){\* 超过实数范围 ，报错 \*}

then begin

error(33);

resulttype := notyp{\* 设置返回值无类型 \*}

end

else if ( a = notyp ) or ( b = notyp )

then resulttype := notyp {\* 设置返回值无类型 \*}

else if a = ints

then if b = ints

then resulttype := ints{\* 都是整型，结果也是整型 \*}

else begin

resulttype := reals;

emit1(26,1) {\* a转换为浮点数 \*}

end

else begin

resulttype := reals;

if b = ints

then emit1(26,0) {\* b转换为浮点数 \*}

end

end { resulttype } ;

procedure expression( fsys: symset; var x: item );{\* 分析处理表达式，由参数（x）返回表达式结果类型，用于检查表达式中各部分的类型是否匹配，若匹配则生成相应Pcode指令 \*}

var y : item;{\* 类型（typ）和地址（ref） \*}

op : symbol;{\* 枚举类型 \*}

procedure simpleexpression( fsys: symset; var x: item );{\* 处理简单表达式，由参数（x）返回求值结果的类型 \*}

var y : item;

op : symbol;

procedure term( fsys: symset; var x: item );{\* 处理项，由参数返回结果类型 \*}

var y : item;

op : symbol;

procedure factor( fsys: symset; var x: item );{\* 处理因子，由参数返回结果类型 \*}

var i,f : integer;

procedure standfct( n: integer );{\* 处理标准函数调用 \*}

var ts : typset;{\* 集合类型 \*}

begin {\* standard function no. n \*}

if sy = lparent{\* 是’(’继续读取，否则报错 \*}

then insymbol

else error(9);

if n < 17

then begin

expression( fsys+[rparent], x );{\* 递归调用处理表达式的分程序来处理参数 \*}

case n of

{ abs, sqr } 0,2: begin{\* 求绝对值或求平方 \*}

ts := [ints, reals];{\* 参数类型为整型或实型 \*}

tab[i].typ := x.typ;{\* 设置函数返回值的类型 \*}

if x.typ = reals{\* 如果返回值的类型是实型，则函数编号加一 \*}

then n := n + 1

end;

{ odd, chr } 4,5: ts := [ints];{\* 判断奇数偶数、字符转化为ASCII码；类型为整型 \*}

{ odr } 6: ts := [ints,bools,chars];{\* 将ASCII码转化为字符；参数类型为整型，布尔型或字符型 \*}

{ succ,pred } 7,8 : begin

ts := [ints, bools,chars];{\* 求后继和前序，参数类型为整型，布尔型或字符型 \*}

tab[i].typ := x.typ

end;

{ round,trunc } 9,10,11,12,13,14,15,16:{\* 各种数学运算 \*}

{ sin,cos,... } begin

ts := [ints,reals];{\* 参数类型为整型或实型 \*}

if x.typ = ints{\* 如果是整型则转化为实型 \*}

then emit1(26,0)

end;

end; { case }

if x.typ in ts{\* 返回值的类型在类型集合中 \*}

then emit1(8,n){\* 产生标准函数（函数编号由n指定） \*}

else if x.typ <> notyp{\* 如果返回值的类型不在类型集合中x保存返回值的类型，如果返回值有类型，报错 \*}

then error(48);

end

else begin { n in [17,18] }{\* 如果函数编号是17或18，则判断输入是否结束 \*}

if sy <> ident{\* 读到的不是普通标识符则报错 \*}

then error(2)

else if id <> 'input '{\* 如果名字不是input就报错 \*}

then error(0)

else insymbol;

emit1(8,n);{\* 生成标准函数 \*}

end;

x.typ := tab[i].typ;{\* 返回值的类型保存在x中 \*}

if sy = rparent{\* 如果是’)’，继续读取否则报错 \*}

then insymbol

else error(4)

end {\* standfct \*} ;

begin {\* factor \*}

x.typ := notyp;{\* 初始化返回值的类型为无类型 \*}

x.ref := 0;

test( facbegsys, fsys,58 );{\* 检测当前符号是不是属于因子开始的标志 \*}

while sy in facbegsys do

begin

if sy = ident{\* 读到普通标识符 \*}

then begin

i := loc(id);{\* 用i记录标识符在符号表中的位置 \*}

insymbol;

with tab[i] do

case obj of

konstant: begin{\* 如果种类是常量，则设置ref为0 \*}

x.typ := typ;

x.ref := 0;

if x.typ = reals{\* 若是实数则生成装入实数的Pcode \*}

then emit1(25,adr)

else emit1(24,adr) {\* 否则生成装入字面变量的Pcode \*}

end;

vvariable:begin{\* 如果种类是变量，则设置ref为符号表中ref的值 \*}

x.typ := typ;

x.ref := ref;

if sy in [lbrack, lparent,period]{\* 如果新读入的符号是’[’’(’’.’之一 \*}

then begin

if normal{\* 参数是值形参或其他普通参数 \*}

then f := 0{\* 取地址并放到栈顶 \*}

else f := 1;{\* 取值并放到栈顶 \*}

emit2(f,lev,adr);{\* 生成Pcode指令，lev为层次，adr为参数的相对地址；lev是静态层次，adr是动态地址，如何得到？ \*}

selector(fsys,x);{\* 处理数组下标变量或记录成员变量 \*}

if x.typ in stantyps{\* 如果x是标准类型 \*}

then emit(34){\* 取栈顶单元内容为地址的单元内容，如果是标准类型说明是值形参，因此取地址 \*}

end

else begin{\* 如果新读入的符号不是’[’’(’’.’之一 \*}

if x.typ in stantyps{\* 如果x是标准类型 \*}

then if normal{\* 如果是标准类型且为值形参，把值装入栈顶 \*}

then f := 1

else f := 2{\* 如果是标准类型且为变量形参，间接装入值 \*}

else if normal{\* 如果不是标准类型且为值形参，把变量地址装入栈顶 \*}

then f := 0

else f := 1;{\* 如果不是标准类型且为变量形参，把值装入栈顶 \*}

emit2(f,lev,adr){\* 生成Pcode指令 \*}

end

end;

typel,prozedure: error(44);{\* 因子只能是常量、变量或函数，因为因子只能是数组的其中一项或者记录的一个域，因此其种类不是数组或记录 \*}

funktion: begin{\* 如果种类是函数，进行函数调用 \*}

x.typ := typ;

if lev <> 0{\* 如果层次不是0，就不是标准函数；标准函数都存在符号表第0层中 \*}

then call(fsys,i){\* 调用非标准函数或过程，i是函数或过程在符号表中的位置 \*}

else standfct(adr){\* 如果层次是0，则调用标准函数 \*}

end

end { case,with }

end

else if sy in [ charcon,intcon,realcon ]{\* 如果是字符常量、整数常量或实数常量；单个常量作为因子 \*}

then begin

if sy = realcon{\* 如果是实数 \*}

then begin

x.typ := reals;{\* 返回值类型是实数（就是本身） \*}

enterreal(rnum);{\* 登录实常量表 \*}

emit1(25,c1){\* 生成Pcode指令，将实数装入栈顶 \*}

end

else begin

if sy = charcon{\* 如果是字符常量 \*}

then x.typ := chars{\* 返回值类型是字符 \*}

else x.typ := ints;{\* 如果不是字符常量则只能是整数常量，返回值类型为整数 \*}

emit1(24,inum){\* 生成Pcode指令，将字面常量装入栈顶 \*}

end;

x.ref := 0;{\* 常量的ref设置为0 \*}

insymbol

end

else if sy = lparent{\* 如果读到’(’，说明括号中的整个表达式作为一个因子 \*}

then begin

insymbol;

expression(fsys + [rparent],x);{\* 递归处理括号中的表达式 \*}

if sy = rparent{\* 如果读到’)’，说明括号完，否则报错 \*}

then insymbol

else error(4)

end

else if sy = notsy{\* 符号的类型未定义，说明是’NOT’标志 \*}

then begin

insymbol;

factor(fsys,x);{\* 处理’NOT’标志后面的表达式 \*}

if x.typ = bools{\* 如果返回的类型是布尔型 \*}

then emit(35){\* 生成逻辑非 \*}

else if x.typ <> notyp{\* 如果x不是布尔型或无类型（接着一个’NOT’）,报错 \*}

then error(32)

end;

test(fsys,facbegsys,6){\* 检测读到的符号是否合法 \*}

end { while }

end { factor };

begin { term }{\* 处理项，由参数返回结果类型 \*}

factor( fsys + [times,rdiv,idiv,imod,andsy],x);{\* 项的第一个部分一定是因子，因此调用因子处理程序 \*}

while sy in [times,rdiv,idiv,imod,andsy] do{\* 如果当前读到的操作符是’\*’’/’’DIV’’MOD’’AND’之一，进入循环（可能存在连续的计算操作） \*}

begin

op := sy;{\* 记录当前运算符 \*}

insymbol;

factor(fsys+[times,rdiv,idiv,imod,andsy],y );{\* 读取第二个因子，类型记录在y中 \*}

if op = times{\* 如果运算符是’\*’ \*}

then begin

x.typ := resulttype(x.typ, y.typ);{\* 获取乘法运算结果的类型 \*}

case x.typ of

notyp: ;

ints : emit(57);{\* 生成Pcode指令整型乘 \*}

reals: emit(60);{\* 生成Pcode指令实型乘 \*}

end

end

else if op = rdiv{\* 如果运算符是’/’ \*}

then begin

if x.typ = ints{\* 如果第一个操作数是整数 \*}

then begin

emit1(26,1);{\* 第一个操作数转换为浮点数 \*}

x.typ := reals;{\* 计算结果的类型是实数 \*}

end;

if y.typ = ints{\* 如果第二个操作数是整数 \*}

then begin

emit1(26,0);{\* 第二个操作数转换为浮点数 \*}

y.typ := reals;{\* 计算结果的类型是实数 \*}

end;

if (x.typ = reals) and (y.typ = reals){\* 如果都是实数 \*}

then emit(61){\* 生成Pcode指令实型除 \*}

else begin

if( x.typ <> notyp ) and (y.typ <> notyp){\* 如果两个操作数都不是未定义类型，报错 \*}

then error(33);

x.typ := notyp{\* 不生成计算指令，设置返回值的类型为无类型 \*}

end

end

else if op = andsy{\* 如果运算符是’+’ \*}

then begin

if( x.typ = bools )and(y.typ = bools){\* 如果x和y都是布尔类型 \*}

then emit(56){\* 生成Pcode指令逻辑与 \*}

else begin

if( x.typ <> notyp ) and (y.typ <> notyp) {\* 如果两个操作数都不是未定义类型，报错 \*}

then error(32);

x.typ := notyp{\* 不生成计算指令，设置返回值的类型为无类型 \*}

end

end

else begin {\* op in [idiv,imod] \*}{\* 如果是’DIV’或’MOD’ \*}

if (x.typ = ints) and (y.typ = ints){\* 都是整型 \*}

then if op = idiv

then emit(58){\* 生成Pcode指令整型除 \*}

else emit(59){\* 生成Pcode指令取模 \*}

else begin

if ( x.typ <> notyp ) and (y.typ <> notyp) {\* 如果两个操作数都不是未定义类型，报错 \*}

then error(34);

x.typ := notyp{\* 不生成计算指令，设置返回值的类型为无类型 \*}

end

end

end { while }

end { term };

begin { simpleexpression }{\* 开始处理简单表达式 \*}

if sy in [plus,minus]{\* 如果是’+’或者’-’ \*}

then begin

op := sy;{\* 记录操作符 \*}

insymbol;

term( fsys+[plus,minus],x);{\* 操作符后一定是项，调用项的处理程序 \*}

if x.typ > reals{\* 如果类型大于实型，报错；为什么类型大于实型？ \*}

then error(33)

else if op = minus{\* 如果操作符是’-’ \*}

then emit(36){\* 生成Pcode指令求负 \*}

end

else term(fsys+[plus,minus,orsy],x);{\* 如果前面没有正负号，则直接处理项 \*}

while sy in [plus,minus,orsy] do{\* 循环处理后面项之间的’+’’-’’OR’操作 \*}

begin

op := sy;{\* 记录操作符 \*}

insymbol;

term(fsys+[plus,minus,orsy],y);{\* 处理项 \*}

if op = orsy{\* 如果操作符是’OR’ \*}

then begin

if ( x.typ = bools )and(y.typ = bools){\* 如果操作数都是bool型 \*}

then emit(51){\* 生成Pcode指令逻辑或 \*}

else begin

if( x.typ <> notyp) and (y.typ <> notyp){\* 如果两个操作数都不是未定义类型，报错 \*}

then error(32);

x.typ := notyp{\* 不生成计算指令，设置返回值的类型为无类型 \*}

end

end

else begin{\* 如果不是’OR’ \*}

x.typ := resulttype(x.typ,y.typ);{\* 获得运算结果的类型，加减操作相同 \*}

case x.typ of

notyp: ;{\* 如果结果的类型为空，不生成任何计算指令 \*}

ints: if op = plus

then emit(52){\* 生成Pcode指令整型加 \*}

else emit(53);{\* 生成Pcode指令整型减 \*}

reals:if op = plus

then emit(54) {\* 生成Pcode指令实型加 \*}

else emit(55) {\* 生成Pcode指令实型减 \*}

end { case }

end

end { while }

end { simpleexpression };

begin { expression }{\* 开始处理完整表达式 \*}

simpleexpression(fsys+[eql,neq,lss,leq,gtr,geq],x);{\* 第一部分是简单表达式，进行处理 \*}

if sy in [ eql,neq,lss,leq,gtr,geq]{\* 如果读到的符号是’=’’<>’’<’’<=’’>’’>=’之一；如果读到的符号不是上述比较符号，表达式其实就是简单表达式x，返回类型也就是x的类型；如果读到的符号是上述比较符号，则返回值的类型是布尔型 \*}

then begin

op := sy;{\* 记录操作符 \*}

insymbol;

simpleexpression(fsys,y);{\* 处理下一个简单表达式 \*}

if(x.typ in [notyp,ints,bools,chars]) and (x.typ = y.typ){\* 两个操作数类型相同且类型为无类型、整型、布尔型、字符型之一 \*}

then case op of

eql: emit(45);{\* 生成Pcode指令整型相等比较 \*}

neq: emit(46);{\* 生成Pcode指令整型不等比较 \*}

lss: emit(47);{\* 生成Pcode指令整型小于比较 \*}

leq: emit(48);{\* 生成Pcode指令整型小于等于比较 \*}

gtr: emit(49);{\* 生成Pcode指令整型大于比较 \*}

geq: emit(50);{\* 生成Pcode指令整型大于等于比较 \*}

end

else begin

if x.typ = ints{\* 如果两个操作数的类型不同且x的类型为整型 \*}

then begin

x.typ := reals;{\* 令x的类型为实型 \*}

emit1(26,1) {\* 生成Pcode指令把x的类型转换为浮点数 \*}

end

else if y.typ = ints{\* 如果两个操作数的类型不同、x不为整型、y为整型 \*}

then begin

y.typ := reals;{\* 令y的类型为实型 \*}

emit1(26,0) {\* 生成Pcode指令把y的类型转换为浮点数 \*}

end;

if ( x.typ = reals)and(y.typ=reals){\* 如果原本类型不相同的两个数经过转换之后都为实型 \*}

then case op of

eql: emit(39); {\* 生成Pcode指令实型等于比较 \*}

neq: emit(40); {\* 生成Pcode指令实型不等比较 \*}

lss: emit(41); {\* 生成Pcode指令实型小于比较 \*}

leq: emit(42); {\* 生成Pcode指令实型小于等于比较 \*}

gtr: emit(43); {\* 生成Pcode指令实型大于比较 \*}

geq: emit(44); {\* 生成Pcode指令实型大于等于比较 \*}

end

else error(35){\* 若类型转换后不全为实型，报错 \*}

end;

x.typ := bools{\* 有比较操作的表达式的返回值类型设置为布尔型 \*}

end

end { expression };

procedure assignment( lv, ad: integer );{\* 处理赋值语句 \*}

var x,y: item;{ typ，ref }

f : integer;

begin { tab[i].obj in [variable,prozedure] }{\* 只有变量类型或者过程类型可以被赋值 \*}

x.typ := tab[i].typ;{\* 保存被赋值变量的类型 \*}

x.ref := tab[i].ref;{\* 保存被赋值变量的指针 \*}

if tab[i].normal{\* 如果当前符号是值形参或其他参数，把参数的地址装入栈顶 \*}

then f := 0

else f := 1;{\* 如果是变量形参，把变量形参的值装入栈顶 \*}

emit2(f,lv,ad);{\* 生成Pcode指令 \*}

if sy in [lbrack,lparent,period]{\* 如果当前符号是’[’’(’’.’之一 \*}

then selector([becomes,eql]+fsys,x);{\* 处理数组下标变量或结构成员变量 \*}

if sy = becomes{\* 如果当前符号是’:=’ \*}

then insymbol

else begin{\* 如果不是，报错 \*}

error(51);

if sy = eql{\* 如果当前符号是’=’，容错，继续读取 \*}

then insymbol

end;

expression(fsys,y);{\* 赋值符号后面接着表达式，处理表达式 \*}

if x.typ = y.typ{\* 赋值符号左右两边类型相同 \*}

then if x.typ in stantyps{\* 如果x是标准类型，可以直接赋值 \*}

then emit(38){\* 生成Pcode指令，将栈顶内容存入次栈顶 \*}

else if x.ref <> y.ref{\* 如果赋值符号左右两边的指针不同，报错 \*}

then error(46)

else if x.typ = arrays{\* 如果x的类型是数组 \*}

then emit1(23,atab[x.ref].size){\* 把数组表中的项拷贝到栈顶 \*}

else emit1(23,btab[x.ref].vsize){\* 把该记录在S栈中的局部变量、参数、内务信息区拷贝到栈顶 \*}

else if(x.typ = reals )and (y.typ = ints){\* 如果x是实型而y是整型 \*}

then begin

emit1(26,0);{\* 生成Pcode指令把y转换为浮点数 \*}

emit(38){\* 生成Pcode指令，写字符 \*}

end

else if ( x.typ <> notyp ) and ( y.typ <> notyp ){\* 如果x和y都不是未知类型，报错 \*}

then error(46)

end { assignment };

procedure compoundstatement;{\* 处理复合语句 \*}

begin

insymbol;

statement([semicolon,endsy]+fsys);{\* 递归处理语句 \*}

while sy in [semicolon]+statbegsys do{\* 如果读到’;’或标准类型集合 \*}

begin

if sy = semicolon{\* 如果是’;’，继续读 \*}

then insymbol

else error(14);{\* 否则报错 \*}

statement([semicolon,endsy]+fsys){\* 递归处理分号之后的语句 \*}

end;

if sy = endsy{\* 如果读到’END’，继续读 \*}

then insymbol

else error(57){\* 没有’END’，报错 \*}

end {\* compoundstatement \*};

procedure ifstatement;{\* 处理if语句 \*}

var x : item;

lc1,lc2: integer;

begin

insymbol;

expression( fsys+[thensy,dosy],x);{\* pascal中if语句后面直接跟表达式，因此要调用表达式处理 \*}

if not ( x.typ in [bools,notyp]){\* 如果读到的表达式不是布尔型或未知类型，报错 \*}

then error(17);

lc1 := lc;{\* lc1保存当前代码读取位置 \*}

emit(11); { jmpc }{\* 生成Pcode代码，若栈顶内容为假则转移到y（这里的y是else语句开始的地方） \*}

if sy = thensy{\* 如果读到的是’then’，继续读 \*}

then insymbol

else begin

error(52);{\* 否则报错 \*}

if sy = dosy{\* 如果读到的是’do’，容错，继续读 \*}

then insymbol

end;

statement( fsys+[elsesy]);{\* 递归处理语句 \*}

if sy = elsesy{\* 如果读到’else’ \*}

then begin

insymbol;

lc2 := lc;{\* lc2保存当前代码读取位置 \*}

emit(10);{\* 生成Pcode指令，无条件转移到y（如果能读到这里说明上面的if为真且已经执行，因此跳转到else语句结束的位置） \*}

code[lc1].y := lc;{\* 设置lc1位置Pcode代码的y为当前代码读取位置 \*}

statement(fsys);{\* 递归处理else语句 \*}

code[lc2].y := lc{\* 设置lc2位置Pcode代码的y为当前代码读取位置 \*}

end

else code[lc1].y := lc{\* 如果没有else，则设置lc1位置的Pcode代码的y为当前代码读取位置，if执行完后直接跳转到这 \*}

end { ifstatement };

procedure casestatement;{\* 处理case语句 \*}

var x : item;

i,j,k,lc1 : integer;{\* i：casetab的项数，j：exittab的项数 \*}

casetab : array[1..csmax]of

packed record

val,lc : index

end;

exittab : array[1..csmax] of integer;

procedure caselabel;{\* 处理case语句中的标号，将各标号对应的目标代码入口地址填入casetab表中，并检查标号有无重复定义 \*}

var lab : conrec;{\* tp，i或r \*}

k : integer;

begin

constant( fsys+[comma,colon],lab );{\* 处理常量，返回的常量值和类型保存在lab中 \*}

if lab.tp <> x.typ{\* 如果返回的常量类型与x的类型不同，报错 \*}

then error(47)

else if i = csmax{\* 如果超出casetab表范围，报错，打印表格溢出信息 \*}

then fatal(6)

else begin

i := i+1;

k := 0;

casetab[i].val := lab.i{\* 将新的常量值保存到表中 \*};

casetab[i].lc := lc;{\* 保存当前代码的读取位置 \*}

repeat

k := k+1

until casetab[k].val = lab.i;{\* 查找casetab表中元素是否与当前常量相同，避免重复定义 \*}

if k < i{\* 找到，报错 \*}

then error(1); {\* multiple definition \*}

end

end { caselabel };

procedure onecase;{\* 处理case语句的一个分支 \*}

begin

if sy in constbegsys{\* 如果读到的符号是常量开始的符号 \*}

then begin

caselabel;{\* 处理case语句中的标号 \*}

while sy = comma do{\* 如果读到’,’，说明还有其他标签 \*}

begin

insymbol;

caselabel{\* 继续读取标签 \*}

end;

if sy = colon{\* 如果读到’:’ \*}

then insymbol

else error(5);{\* 若没有则报错 \*}

statement([semicolon,endsy]+fsys);{\* 递归处理语句 \*}

j := j+1;{\* 增加exittab项数 \*}

exittab[j] := lc;{\* 将当前位置记录在exittab中 \*}

emit(10){\* 生成Pcode指令，无条件转移到y \*}

end

end { onecase };

begin { casestatement }{\* 开始处理case语句 \*}

insymbol;

i := 0;{\* 初始化casetab的项数 \*}

j := 0;{\* 初始化exittab的项数 \*}

expression( fsys + [ofsy,comma,colon],x );{\* 处理表达式，获得表达式的值 \*}

if not( x.typ in [ints,bools,chars,notyp ]){\* 如果表达式不是这些类型，case语句无法处理，报错 \*}

then error(23);

lc1 := lc;{\* 记录当前代码读取位置 \*}

emit(12); { jmpx }{\* 生成Pcode指令，转移到y，查找情况表 \*}

if sy = ofsy{\* 如果读到’of’ \*}

then insymbol

else error(8);{\* 如果没有读到就报错 \*}

onecase;{\* 处理一个case项 \*}

while sy = semicolon do{\* 如果读到’;’ \*}

begin

insymbol;

onecase{\* 继续处理case项 \*}

end;

code[lc1].y := lc;{\* 把Pcode表中case语句开始位置指令的y值设置为当前读取位置 \*}

for k := 1 to i do{\* 遍历所有标签 \*}

begin

emit1( 13,casetab[k].val);{\* 生成Pcode指令，情况表的登记项；如果值相同 \*}

emit1( 13,casetab[k].lc);{\* 生成Pcode指令，情况表的登记项：就跳转到当前case的处理地址 \*}

end;

emit1(10,0);{\* 生成Pcode指令，无条件转移到y \*}

for k := 1 to j do{\* 遍历所有case项 \*}

code[exittab[k]].y := lc;{\* 所有项的退出位置（y）都设置为当前代码读取位置 \*}

if sy = endsy{\* 读到’end’，继续读 \*}

then insymbol

else error(57){\* 否则报错 \*}

end {\* casestatement \*};

procedure repeatstatement;{\* 处理repeat语句 \*}

var x : item;

lc1: integer;

begin

lc1 := lc;{\* 记录当前代码读取位置 \*}

insymbol;

statement( [semicolon,untilsy]+fsys);{\* 递归处理repeat块内部语句 \*}

while sy in [semicolon]+statbegsys do{\* 如果读到’;’或语句块开始的符号集合 \*}

begin

if sy = semicolon{\* 如果读到分号 \*}

then insymbol

else error(14);{\* 否则报错 \*}

statement([semicolon,untilsy]+fsys){\* 递归处理分号之后的语句 \*}

end;

if sy = untilsy{\* 若读到’until’ \*}

then begin

insymbol;

expression(fsys,x);{\* 处理until内的表达式 \*}

if not(x.typ in [bools,notyp] ){\* 如果表达式的类型不是布尔型或者未知类型，报错 \*}

then error(17);

emit1(11,lc1);{\* 生成Pcode指令，如果until内表达式为假，则跳转到repeat语句块开头 \*}

end

else error(53){\* 如果没读到’until’则报错 \*}

end { repeatstatement };

procedure whilestatement;{\* 处理while语句；pascal语法中while语句块内只能有一条语句，因此不用读到’;’ \*}

var x : item;

lc1,lc2 : integer;

begin

insymbol;

lc1 := lc;{\* 记录当前代码读取位置 \*}

expression( fsys+[dosy],x);{\* 处理while内的表达式 \*}

if not( x.typ in [bools, notyp] ){\* 如果while内表达式的类型不是布尔型或者未知类型，报错 \*}

then error(17);

lc2 := lc;{\* 记录while块代码开始的位置 \*}

emit(11);{\* 生成Pcode指令，如果while表达式栈顶的内容为假，跳转到while块结束 \*}

if sy = dosy{\* 如果读到’do’，继续读 \*}

then insymbol

else error(54);{\* 否则报错 \*}

statement(fsys);{\* 递归处理while块中的语句 \*}

emit1(10,lc1);{\* 无条件转移到y，下一次循环 \*}

code[lc2].y := lc{\* 记录退出循环的位置 \*}

end { whilestatement };

procedure forstatement;{\* 处理for语句；根据pascal语法，for循环块中只能有一条语句，因此不用读取’;’ \*}

var cvt : types;{\* 枚举，计数变量的类型 \*}

x : item;

i,f,lc1,lc2 : integer;

begin

insymbol;

if sy = ident{\* 如果读到普通标识符 \*}

then begin

i := loc(id);{\* 获得当前标识符在符号表中的位置 \*}

insymbol;

if i = 0

then cvt := ints{\* 符号表中没有记录当前标识符，默认计数变量类型为int \*}

else if tab[i].obj = vvariable{\* 如果已经记录计数变量且其种类为variable \*}

then begin

cvt := tab[i].typ;{\* 记录循环变量的类型 \*}

if not tab[i].normal{\* 如果循环变量是变量形参，报错（变量形参不能用于for循环的循环变量） \*}

then error(37)

else emit2(0,tab[i].lev, tab[i].adr );{\* 生成Pcode指令，把循环变量的地址装入S栈 \*}

if not ( cvt in [notyp, ints, bools, chars]){\* 如果循环变量的类类型在上述集合中，报错 \*}

then error(18)

end

else begin{\* 如果已经记录计数变量但其种类不是variable \*}

error(37);

cvt := ints{\* 设循环变量类型为整型 \*}

end

end

else skip([becomes,tosy,downtosy,dosy]+fsys,2);{\* 如果不是标识符，跳过 \*}

if sy = becomes{\* 如果读到’:=’ \*}

then begin

insymbol;

expression( [tosy, downtosy,dosy]+fsys,x);{\* 处理’:=’后面的表达式 \*}

if x.typ <> cvt{\* 如果表达式的类型和循环变量类型不同，报错 \*}

then error(19);

end

else skip([tosy, downtosy,dosy]+fsys,51);{\* 如果不是’:=’，跳过 \*}

f := 14;{\* 增量步长型for循环体的入口测试 \*}

if sy in [tosy,downtosy]{\* 如果读到’to’或’downto’ \*}

then begin

if sy = downtosy{\* 如果读到’downto’ \*}

then f := 16;{\* 减量步长型for循环体的入口测试 \*}

insymbol;

expression([dosy]+fsys,x);{\* 处理终止条件表达式 \*}

if x.typ <> cvt{\* 如果终止条件表达式的类型和循环变量类型不同，报错 \*}

then error(19)

end

else skip([dosy]+fsys,55);{\* 如果不是’to’或’downto’，跳过 \*}

lc1 := lc;{\* 记录for循环开始前的代码位置 \*}

emit(f);{\* 生成Pcode指令，for循环体入口测试 \*}

if sy = dosy{\* 如果读到’do’ \*}

then insymbol

else error(54);{\* 若没读到，报错 \*}

lc2 := lc;{\* 记录处理完for循环语句块开始的代码位置 \*}

statement(fsys);{\* 处理for循环语句块 \*}

emit1(f+1,lc2);{\* 生成Pcode代码，增量/减量for循环的再入口测试 \*}

code[lc1].y := lc{\* 设置for循环开始之前位置的Pcode代码的y值为当前代码位置，用于跳出for循环 \*}

end { forstatement };

procedure standproc( n: integer );{\* 处理标准（输入/输出）过程调用 \*}

var i,f : integer;

x,y : item;

begin

case n of{\* n表示当前标准过程是输入还是输出 \*}

1,2 : begin { read }{\* 如果是1、2则表示标准输入 \*}

if not iflag{\* 如果iflag为false，报错，并重置iflag为true \*}

then begin

error(20);

iflag := true

end;

if sy = lparent{\* 如果读到’(’ \*}

then begin

repeat

insymbol;

if sy <> ident{\* 如果读到的不是普通标识符，报错 \*}

then error(2)

else begin

i := loc(id);{\* 获得当前符号在符号表中的位置 \*}

insymbol;

if i <> 0{\* 如果当前符号在符号表中 \*}

then if tab[i].obj <> vvariable{\* 如果当前符号的种类不是variable，报错 \*}

then error(37)

else begin{\* 如果种类是variable \*}

x.typ := tab[i].typ;{\* 记录该变量的种类 \*}

x.ref := tab[i].ref;{\* 记录改变量的指针 \*}

if tab[i].normal{\* 值形参装地址，变量形参装值 \*}

then f := 0

else f := 1;

emit2(f,tab[i].lev,tab[i].adr);{\* 生成Pcode指令装入栈顶 \*}

if sy in [lbrack,lparent,period]{\* 如果读到’[’,’(’,’.’ \*}

then selector( fsys+[comma,rparent],x);{\* 处理数组下标变量或结构成员变量 \*}

if x.typ in [ints,reals,chars,notyp]{\* 如果读到的要输入值的变量的类型在上述集合中 \*}

then emit1(27,ord(x.typ)){\* 生成Pcode代码，根据变量类型标准读入不同类型的数据 \*}

else error(41){\* 如果要输入值的变量的类型不对则报错 \*}

end

end;

test([comma,rparent],fsys,6);{\* 测试读到的符号是不是’,’或’)’ \*}

until sy <> comma;{\* 如果读到’,’，说明有其他变量需要输入值，循环读取 \*}

if sy = rparent{\* 如果读到’)’，继续读取 \*}

then insymbol

else error(4){\* 否则报错 \*}

end;

if n = 2{\* 如果n为2，说明要读完一行 \*}

then emit(62){\* 生成Pcode指令，读完一行 \*}

end;

3,4 : begin { write }{\* 如果是3、4表示标准输出 \*}

if sy = lparent{\* 如果读到’(’ \*}

then begin

repeat

insymbol;

if sy = stringcon{\* 如果读到字符串常量 \*}

then begin

emit1(24,sleng);{\* 生成Pcode指令，装入字面常量 \*}

emit1(28,inum);{\* 生成Pcode指令，标准输出 \*}

insymbol

end

else begin{\* 如果不是字符串常量，说明要输出标准类型的变量 \*}

expression(fsys+[comma,colon,rparent],x);{\* 处理表达式，得到表达式的类型 \*}

if not( x.typ in stantyps ){\* 如果要输出的变量x的类型不是标准类型，报错 \*}

then error(41);

if sy = colon{\* 如果读到’:’ \*}

then begin

insymbol;

expression( fsys+[comma,colon,rparent],y);{\* 处理表达式，得到表达式的类型 \*}

if y.typ <> ints{\* 如果表达式的类型不是整型，报错 \*}

then error(43);

if sy = colon{\* 如果再次读到’:’ \*}

then begin

if x.typ <> reals{\* 如果要输出的变量x的类型不是实型，报错（因为第二个’:’后面代表小数位数） \*}

then error(42);

insymbol;

expression(fsys+[comma,rparent],y);{\* 读取表达式，获得表达式类型 \*}

if y.typ <> ints{\* 如果第二个’:’后面的表达式不是整型，报错 \*}

then error(43);

emit(37){\* 生成Pcode指令，标准输出给定了域宽的实数 \*}

end

else emit1(30,ord(x.typ)){\* 生成Pcode指令，标准输出给定了域宽的变量 \*}

end

else emit1(29,ord(x.typ)){\* 生成Pcode指令，标准输出没有显示定义域宽的变量 \*}

end

until sy <> comma;{\* 如果读到’,’说明有其他变量要输出，循环输出 \*}

if sy = rparent{\* 如果读到’)’，继续读 \*}

then insymbol

else error(4){\* 否则报错 \*}

end;

if n = 4{\* 如果n是4，说明换行写 \*}

then emit(63){\* 生成Pcode指令，换行写 \*}

end; { write }

end { case };

end { standproc } ;

begin { statement }{\* 开始处理语句 \*}

if sy in statbegsys+[ident]{\* 如果读到语句开始符号或者普通标识符 \*}

then case sy of

ident : begin{\* 如果读到普通标识符 \*}

i := loc(id);{\* 获取当前符号在符号表中的位置 \*}

insymbol;

if i <> 0{\* 如果在符号表中找到当前符号 \*}

then case tab[i].obj of

konstant,typel : error(45);{\* 如果其种类是constant或type类型，报错，因为不能出现在语句开头 \*}

vvariable: assignment( tab[i].lev,tab[i].adr);{\* 是variable类型，说明是赋值语句 \*}

prozedure: if tab[i].lev <> 0

then call(fsys,i){\* 是procedure类型且不是标准过程，调用非标准过程 \*}

else standproc(tab[i].adr);{\* 是预定义函数且是标准输入/输出，调用标准输入/输出 \*}

funktion: if tab[i].ref = display[level]{\* 如果符号表中记录的指针与display表中当前层数的索引一致 \*}

then assignment(tab[i].lev+1,0){\* 给函数赋值，层数加一是因为符号表中存储的函数的层次比函数中的变量的层次高一层，因为函数名定义在外层 \*}

else error(45){\* 如果不一致则报错 \*}

end { case }

end;

beginsy : compoundstatement;{\* 如果读到’begin’，说明是复合语句 \*}

ifsy : ifstatement;{\* 如果读到’if’，说明是’if’语句 \*}

casesy : casestatement;{\* 如果读到’case’，说明是’case’语句 \*}

whilesy : whilestatement;{\* 如果是’while’，说明是’while’语句 \*}

repeatsy: repeatstatement;{\* 如果读到’repeat’，说明是’repeat’语句 \*}

forsy : forstatement;{\* 如果读到’for’，说明是’for’语句 \*}

end; { case }

test( fsys, [],14);{\* 检查当前读到的符号是否合法 \*}

end { statement };

begin { block }{\* 开始处理分程序；procedure block( fsys: symset; isfun: boolean; level: integer ) \*}

dx := 5;{\* 现在运行栈S的数据区开头留出5个单元作为内务信息区 \*}

prt := t;{\* 记录当前符号表栈顶指针 \*}

if level > lmax{\* 如果当前层数大于最大层数限制，报错，打印层溢出信息 \*}

then fatal(5);

test([lparent,colon,semicolon],fsys,14);{\* 测试读到的符号是不是’(’’:’’;’其中之一 \*}

enterblock;{\* 登录到分程序表btab \*}

prb := b;{\* 记录当前分程序表栈顶指针 \*}

display[level] := b;{\* 建立当前层到btab表的索引 \*}

tab[prt].typ := notyp;{\* 设置符号表中当前分程序的类型为未知类型 \*}

tab[prt].ref := prb;{\* 设置符号表中当前分程序的索引是当前分程序在btab的指针 \*}

if ( sy = lparent ) and ( level > 1 ){\* 如果读到’(’且层次大于一（不是主程序，主程序层次为1） \*}

then parameterlist;{\* 处理分程序声明中的形参表，将形参信息记录到符号表中 \*}

btab[prb].lastpar := t;{\* 因为参数已读完，因此设置当前符号表栈顶指针为当前分程序最后一个参数在符号表中的位置 \*}

btab[prb].psize := dx;{\* 记录当前分程序和其参数在S栈的内务信息区所占的存储单元数：？ \*}

if isfun{\* 当前分程序如果是函数 \*}

then if sy = colon{\* 如果读到’:’ \*}

then begin

insymbol; {\* function type \*}{\* 函数类型标识符 \*}

if sy = ident{\* 如果读到普通标识符 \*}

then begin

x := loc(id);{\* 获取标识符在符号表中的地址 \*}

insymbol;

if x <> 0{\* 如果符号表中存在该标识符 \*}

then if tab[x].typ in stantyps{\* 如果该标识符的类型是标准类型 \*}

then tab[prt].typ := tab[x].typ{\* 设符号表中当前分程序的类型为标识符的类型 \*}

else error(15){\* 如果不是标准类型，报错 \*}

end

else skip( [semicolon]+fsys,2 ){\* 如果不是普通标识符，跳过 \*}

end

else error(5);{\* 如果不是冒号，报错 \*}

if sy = semicolon{\* 如果读到’;’ \*}

then insymbol

else error(14);{\* 如果没有’;’则报错 \*}

repeat

if sy = constsy{\* 如果读到’const’ \*}

then constdec;{\* 处理常量定义，将常量名和其相应信息填入符号表 \*}

if sy = typesy{\* 如果读到’type’ \*}

then typedeclaration;{\* 处理类型定义，将类型名和其相应信息填入符号表 \*}

if sy = varsy{\* 如果读到’var’ \*}

then variabledeclaration;{\* 处理变量定义，将类型名和其相应信息填入符号表 \*}

btab[prb].vsize := dx;{\* 这时已经读完局部变量，把参数、内务信息区再加上局部变量的在S栈中的总大小记录在vsize中 \*}

while sy in [procsy,funcsy] do{\* 如果读到’procedure’或’function’ \*}

procdeclaration;{\* 处理过程或函数说明，递归调用处理分程序 \*}

test([beginsy],blockbegsys+statbegsys,56){\* 已经处理完过程或函数定义，此时读到的符号应该是’begin’ \*}

until sy in statbegsys;{\* 当符号是语句开头时停止循环 \*}

tab[prt].adr := lc;{\* 设置符号表当前分程序的adr为当前Pcode表指令位置（函数和过程的adr域是相应目标代码的入口地址） \*}

insymbol;

statement([semicolon,endsy]+fsys);{\* 处理分程序中的语句 \*}

while sy in [semicolon]+statbegsys do{\* 如果读到的符号是’;’或者语句开头 \*}

begin

if sy = semicolon{\* 如果读到’;’，继续读 \*}

then insymbol

else error(14);{\* 否则报错 \*}

statement([semicolon,endsy]+fsys);{\* 继续处理分号后面的语句 \*}

end;

if sy = endsy{\* 如果读到’end’，继续读 \*}

then insymbol

else error(57);{\* 否侧报错 \*}

test( fsys+[period],[],6 ){\* 测试当前符号是否合法；主程序结束的时候要有个’.’，因此把’.’加入测试符号 \*}

end { block };

procedure interpret;{\* P代码解释执行程序 \*}

var ir : order ; { instruction buffer }{\* 当前的指令 \*}

pc : integer; { program counter }{\* 指令寄存器 \*}

t : integer; { top stack index }{\* S栈顶指针 \*}

b : integer; { base index }{\* 基地址的指针 \*}

h1,h2,h3: integer;

lncnt,ocnt,blkcnt,chrcnt: integer; { counters }{\* 计数器 \*}

ps : ( run,fin,caschk,divchk,inxchk,stkchk,linchk,lngchk,redchk );{\* 错误信息（枚举类型） \*}

fld: array [1..4] of integer; { default field widths }{\* 默认宽度 \*}

display : array[0..lmax] of integer;{\* 这里的display是各层次嵌套分程序在S栈中的基地址索引表 \*}

s : array[1..stacksize] of { blockmark: }{\* S栈 \*}

record

case cn : types of { s[b+0] = fct result }{\* 存放函数结果，根据函数结果的类型决定其他域的值 \*}

ints : (i: integer ); { s[b+1] = return adr }{\* RA返回地址 \*}

reals :(r: real ); { s[b+2] = static link }{\* SL静态链 \*}

bools :(b: boolean ); { s[b+3] = dynamic link }{\* DL动态链 \*}

chars :(c: char ) { s[b+4] = table index }{\* 过程或函数名在tab表中的位置 \*}

end;

procedure dump;{\* 程序运行时，卸出打印现场剖析信息（display，tab、btab及运行栈的内容） \*}

var p,h3 : integer;

begin

h3 := tab[h2].lev;{\* 获得当前函数或过程的层次，按层打印信息 \*}

writeln(psout);

writeln(psout);

writeln(psout,' calling ', tab[h2].name );{\* 打印调用的函数或过程名 \*}

writeln(psout,' level ',h3:4);{\* 打印层次 \*}

writeln(psout,' start of code ',pc:4);{\* 打印指令寄存器的位置 \*}

writeln(psout);

writeln(psout);

writeln(psout,' contents of display ');{\* 打印display表的内容 \*}

writeln(psout);

for p := h3 downto 0 do

writeln(psout,p:4,display[p]:6);{\* 打印每个层次在S栈的基地址索引 \*}

writeln(psout);

writeln(psout);

writeln(psout,' top of stack ',t:4,' frame base ':14,b:4);{\* 打印S栈顶指针位置以及分程序基地址的指针位置 \*}

writeln(psout);

writeln(psout);

writeln(psout,' stack contents ':20);{\* 打印运行栈的内容 \*}

writeln(psout);

for p := t downto 1 do

writeln( psout, p:14, s[p].i:8);打印{\* 遍历运行栈，打印位置以及内务信息：？ \*}

writeln(psout,'< = = = >':22)

end; { dump }

procedure inter0;{\* 根据不同的Pcode编号类执行不同的操作 \*}

begin

case ir.f of{\* ir.f：当前的指令的编号 \*}

0 : begin { load addrss }{\* 把地址装入S栈顶 \*}

t := t + 1;{\* S栈顶指针 \*}

if t > stacksize{\* 如果超过S栈范围 \*}

then ps := stkchk{\* 生成错误类型 \*}

else s[t].i := display[ir.x]+ir.y{\* 若没有超过，S栈当前位置记录的值为display表中的基地址x对应的数加上偏移y \*}

end;

1 : begin { load value }{\* 把值装入S栈顶 \*}

t := t + 1;

if t > stacksize

then ps := stkchk

else s[t] := s[display[ir.x]+ir.y]{\* 若没有超过，S栈当前位置记录的值为基地址x加上偏移y对应的S栈中的值 \*}

end;

2 : begin { load indirect }{\* 间接装入 \*}

t := t + 1;

if t > stacksize

then ps := stkchk

else s[t] := s[s[display[ir.x]+ir.y].i]{\* 若没有超过，间接装入（描述有困难） \*}

end;

3 : begin { update display }{\* 更新display \*}

h1 := ir.y;{\* 获得当前display区的信息，h1和h2分别为最大索引和最小索引 \*}

h2 := ir.x;

h3 := b;

repeat

display[h1] := h3;{\* 反复刷新display区的值，直到h1等于h2为止 \*}

h1 := h1-1;

h3 := s[h3+2].i

until h1 = h2

end;

8 : case ir.y of{\* 标准函数 \*}

0 : s[t].i := abs(s[t].i);{\* 整数绝对值 \*}

1 : s[t].r := abs(s[t].r);{\* 实数绝对值 \*}

2 : s[t].i := sqr(s[t].i);{\* 整数平方 \*}

3 : s[t].r := sqr(s[t].r);{\* 实数平方 \*}

4 : s[t].b := odd(s[t].i);{\* 整数判断奇偶 \*}

5 : s[t].c := chr(s[t].i);{\* ASCII码转字符 \*}

6 : s[t].i := ord(s[t].c);{\* 字符转ASCII码 \*}

7 : s[t].c := succ(s[t].c);{\* 求字符的后继字符 \*}

8 : s[t].c := pred(s[t].c);{\* 求字符的前序字符 \*}

9 : s[t].i := round(s[t].r);{\* 四舍五入 \*}

10 : s[t].i := trunc(s[t].r);{\* 求整数部分 \*}

11 : s[t].r := sin(s[t].r);{\* 求正弦 \*}

12 : s[t].r := cos(s[t].r);{\* 求余弦 \*}

13 : s[t].r := exp(s[t].r);{\* 求e的x次方 \*}

14 : s[t].r := ln(s[t].r);{\* 求对数 \*}

15 : s[t].r := sqrt(s[t].r);{\* 实数开方 \*}

16 : s[t].r := arcTan(s[t].r);{\* 求anctan \*}

17 : begin

t := t+1;

if t > stacksize

then ps := stkchk

else s[t].b := eof(prd){\* 判断输入是否读完 \*}

end;

18 : begin

t := t+1;

if t > stacksize

then ps := stkchk

else s[t].b := eoln(prd){\* 判断一行输入是否读完 \*}

end;

end;

9 : s[t].i := s[t].i + ir.y; {\* offset \*}{\* 将栈顶元素加上y \*}

end { case ir.f }

end; { inter0 }

procedure inter1;

var h3, h4: integer;

begin

case ir.f of

10 : pc := ir.y ; { jump }{\* 无条件转移到y \*}

11 : begin { conditional jump }{\* 如栈顶内容为假，转移到y \*}

if not s[t].b

then pc := ir.y;

t := t - 1{\* S栈退栈 \*}

end;

12 : begin { switch }{\* 转移到y，查找情况表（情况表的登记项由f为13的指令组成） \*}

h1 := s[t].i;

t := t-1;

h2 := ir.y;

h3 := 0;

repeat

if code[h2].f <> 13

then begin

h3 := 1;

ps := caschk{\* 记录错误信息 \*}

end

else if code[h2].y = h1

then begin

h3 := 1;

pc := code[h2+1].y{\* 跳转到code[h2]的下一条指令 \*}

end

else h2 := h2 + 2

until h3 <> 0

end;

14 : begin { for1up }{\* 增量步长型for循环体的入口测试 \*}

h1 := s[t-1].i;

if h1 <= s[t].i

then s[s[t-2].i].i := h1

else begin

t := t - 3;

pc := ir.y

end

end;

15 : begin { for2up }{\* 增量步长型for循环体的再入测试 \*}

h2 := s[t-2].i;

h1 := s[h2].i+1;

if h1 <= s[t].i

then begin

s[h2].i := h1;

pc := ir.y

end

else t := t-3;

end;

16 : begin { for1down }{\* 减量步长型for循环体的入口测试 \*}

h1 := s[t-1].i;

if h1 >= s[t].i

then s[s[t-2].i].i := h1

else begin

pc := ir.y;

t := t - 3

end

end;

17 : begin { for2down }{\* 减量步长型for循环体的再入测试 \*}

h2 := s[t-2].i;

h1 := s[h2].i-1;

if h1 >= s[t].i

then begin

s[h2].i := h1;

pc := ir.y

end

else t := t-3;

end;

18 : begin { mark stack }{\* 标记栈 \*}

h1 := btab[tab[ir.y].ref].vsize;

if t+h1 > stacksize

then ps := stkchk

else begin

t := t+5;{\* 把S栈指针加5，留出“内务信息区” \*}

s[t-1].i := h1-1;{\* 把本分程序在S栈中将要占据的数据区长度减一记录在DL单元中 \*}

s[t].i := ir.y{\* 把实参表达式的值或实参变量地址放到预留的参数单元中 \*}

end

end;

19 : begin { call }{\* 调用用户过程或函数 \*}

h1 := t-ir.y; {\* h1 points to base \*}

h2 := s[h1+4].i; {\* h2 points to tab \*}

h3 := tab[h2].lev;

display[h3+1] := h1;

h4 := s[h1+3].i+h1;

s[h1+1].i := pc;

s[h1+2].i := display[h3];

s[h1+3].i := b;

for h3 := t+1 to h4 do

s[h3].i := 0;

b := h1;

t := h4;

pc := tab[h2].adr;

if stackdump

then dump

end;

end { case }

end; { inter1 }

procedure inter2;

begin

case ir.f of

20 : begin { index1 }{\* 取下标变量地址（元素长度等于1） \*}

h1 := ir.y; { h1 points to atab }

h2 := atab[h1].low;

h3 := s[t].i;

if h3 < h2

then ps := inxchk

else if h3 > atab[h1].high

then ps := inxchk

else begin

t := t-1;

s[t].i := s[t].i+(h3-h2)

end

end;

21 : begin { index }{\* 取下标变量地址 \*}

h1 := ir.y ; { h1 points to atab }

h2 := atab[h1].low;

h3 := s[t].i;

if h3 < h2

then ps := inxchk

else if h3 > atab[h1].high

then ps := inxchk

else begin

t := t-1;

s[t].i := s[t].i + (h3-h2)\*atab[h1].elsize

end

end;

22 : begin { load block }{\* 装入块 \*}

h1 := s[t].i;

t := t-1;

h2 := ir.y+t;

if h2 > stacksize

then ps := stkchk

else while t < h2 do

begin

t := t+1;

s[t] := s[h1];

h1 := h1+1

end

end;

23 : begin { copy block }{\* 复制块 \*}

h1 := s[t-1].i;

h2 := s[t].i;

h3 := h1+ir.y;

while h1 < h3 do

begin

s[h1] := s[h2];

h1 := h1+1;

h2 := h2+1

end;

t := t-2

end;

24 : begin { literal }{\* 装入字面常量 \*}

t := t+1;

if t > stacksize

then ps := stkchk

else s[t].i := ir.y

end;

25 : begin { load real }{\* 装入实数 \*}

t := t+1;

if t > stacksize

then ps := stkchk

else s[t].r := rconst[ir.y]

end;

26 : begin { float }{\* 转换浮点数 \*}

h1 := t-ir.y;

s[h1].r := s[h1].i

end;

27 : begin { read }{\* 读（y域表示类型，1：整型，2：实型，3：字符型） \*}

if eof(prd)

then ps := redchk

else case ir.y of

1 : read(prd, s[s[t].i].i);

2 : read(prd, s[s[t].i].r);

4 : read(prd, s[s[t].i].c);

end;

t := t-1

end;

28 : begin { write string }{\* 写字符 \*}

h1 := s[t].i;

h2 := ir.y;

t := t-1;

chrcnt := chrcnt+h1;

if chrcnt > lineleng

then ps := lngchk;

repeat

write(prr,stab[h2]);

h1 := h1-1;

h2 := h2+1

until h1 = 0

end;

29 : begin { write1 }{\* 写（隐含域宽） \*}

chrcnt := chrcnt + fld[ir.y];

if chrcnt > lineleng

then ps := lngchk

else case ir.y of

1 : write(prr,s[t].i:fld[1]);

2 : write(prr,s[t].r:fld[2]);

3 : if s[t].b

then write('true')

else write('false');

4 : write(prr,chr(s[t].i));

end;

t := t-1

end;

end { case }

end; { inter2 }

procedure inter3;

begin

case ir.f of

30 : begin { write2 }{\* 写（给定域宽） \*}

chrcnt := chrcnt+s[t].i;

if chrcnt > lineleng

then ps := lngchk

else case ir.y of

1 : write(prr,s[t-1].i:s[t].i);

2 : write(prr,s[t-1].r:s[t].i);

3 : if s[t-1].b

then write('true')

else write('false');

end;

t := t-2

end;

31 : ps := fin;{\* 停止 \*}

32 : begin { exit procedure }{\* 退出过程 \*}

t := b-1;

pc := s[b+1].i;

b := s[b+3].i

end;

33 : begin { exit function }{\* 退出函数 \*}

t := b;

pc := s[b+1].i;

b := s[b+3].i

end;

34 : s[t] := s[s[t].i];{\* 取栈顶单元内容为地址的单元内容 \*}

35 : s[t].b := not s[t].b;{\* 逻辑非 \*}

36 : s[t].i := -s[t].i;{\* 求负 \*}

37 : begin{\* 写实数（给定域宽） \*}

chrcnt := chrcnt + s[t-1].i;

if chrcnt > lineleng

then ps := lngchk

else write(prr,s[t-2].r:s[t-1].i:s[t].i);

t := t-3

end;

38 : begin { store }{\* 将栈顶内容存入次栈顶 \*}

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

t := t-2

end;

39 : begin{\* 实型等于比较 \*}

t := t-1;

s[t].b := s[t].r=s[t+1].r

end;

end { case }

end; { inter3 }

procedure inter4;

begin

case ir.f of

40 : begin{\* 实型不等比较 \*}

t := t-1;

s[t].b := s[t].r <> s[t+1].r

end;

41 : begin{\* 实型小于比较 \*}

t := t-1;

s[t].b := s[t].r < s[t+1].r

end;

42 : begin{\* 实型小于等于比较 \*}

t := t-1;

s[t].b := s[t].r <= s[t+1].r

end;

43 : begin{\* 实型大于比较 \*}

t := t-1;

s[t].b := s[t].r > s[t+1].r

end;

44 : begin{\* 实型大于等于比较 \*}

t := t-1;

s[t].b := s[t].r >= s[t+1].r

end;

45 : begin{\* 实型相等比较 \*}

t := t-1;

s[t].b := s[t].i = s[t+1].i

end;

46 : begin{\* 实型不等比较 \*}

t := t-1;

s[t].b := s[t].i <> s[t+1].i

end;

47 : begin{\* 实型小于比较 \*}

t := t-1;

s[t].b := s[t].i < s[t+1].i

end;

48 : begin{\* 实型小于等于比较 \*}

t := t-1;

s[t].b := s[t].i <= s[t+1].i

end;

49 : begin{\* 实型大于比较 \*}

t := t-1;

s[t].b := s[t].i > s[t+1].i

end;

end { case }

end; { inter4 }

procedure inter5;

begin

case ir.f of

50 : begin{\* 实型大于等于比较 \*}

t := t-1;

s[t].b := s[t].i >= s[t+1].i

end;

51 : begin{\* 逻辑或 \*}

t := t-1;

s[t].b := s[t].b or s[t+1].b

end;

52 : begin{\* 整型加 \*}

t := t-1;

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

end;

53 : begin{\* 整型减 \*}

t := t-1;

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

end;

54 : begin{\* 实型加 \*}

t := t-1;

s[t].r := s[t].r+s[t+1].r;

end;

55 : begin{\* 实型减 \*}

t := t-1;

s[t].r := s[t].r-s[t+1].r;

end;

56 : begin{\* 逻辑与 \*}

t := t-1;

s[t].b := s[t].b and s[t+1].b

end;

57 : begin{\* 整型乘 \*}

t := t-1;

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

end;

58 : begin{\* 整型除 \*}

t := t-1;

if s[t+1].i = 0

then ps := divchk

else s[t].i := s[t].i div s[t+1].i

end;

59 : begin{\* 取模 \*}

t := t-1;

if s[t+1].i = 0

then ps := divchk

else s[t].i := s[t].i mod s[t+1].i

end;

end { case }

end; { inter5 }

procedure inter6;

begin

case ir.f of

60 : begin{\* 实型乘 \*}

t := t-1;

s[t].r := s[t].r\*s[t+1].r;

end;

61 : begin{\* 实型除 \*}

t := t-1;

s[t].r := s[t].r/s[t+1].r;

end;

62 : if eof(prd){\* 读完一行 \*}

then ps := redchk

else readln;

63 : begin{\* 换行写 \*}

writeln(prr);

lncnt := lncnt+1;

chrcnt := 0;

if lncnt > linelimit

then ps := linchk

end

end { case };

end; { inter6 }

begin { interpret }

s[1].i := 0;

s[2].i := 0;

s[3].i := -1;

s[4].i := btab[1].last;{\* 值为28 \*}

display[0] := 0;

display[1] := 0;

t := btab[2].vsize-1;{\* 第一个分程序 \*}

b := 0;{\* 初始化基地址指针为0 \*}

pc := tab[s[4].i].adr;{\* pc初始化为符号表第一个符号的索引地址（对于过程名/函数名，adr是过程/函数目标代码的入口地址） \*}

lncnt := 0;

ocnt := 0;

chrcnt := 0;

ps := run;{\* 初始化运行状态 \*}

fld[1] := 10;

fld[2] := 22;

fld[3] := 10;

fld[4] := 1;

repeat

ir := code[pc];{\* 获取当前位置代码 \*}

pc := pc+1;{\* 代码计数器加一 \*}

ocnt := ocnt+1;

case ir.f div 10 of{\* 根据Pcode指令编号执行指令 \*}

0 : inter0;

1 : inter1;

2 : inter2;

3 : inter3;

4 : inter4;

5 : inter5;

6 : inter6;

end; { case }

until ps <> run;{\* 当运行出错时停止 \*}

if ps <> fin{\* 如果当前状态不是读文件 \*}

then begin

writeln(prr);

write(prr, ' halt at', pc :5, ' because of ');{\* 打印出错信息和行数；pc：5？ \*}

case ps of{\* 根据运行状态输出错误信息 \*}

caschk : writeln(prr,'undefined case');

divchk : writeln(prr,'division by 0');

inxchk : writeln(prr,'invalid index');

stkchk : writeln(prr,'storage overflow');

linchk : writeln(prr,'too much output');

lngchk : writeln(prr,'line too long');

redchk : writeln(prr,'reading past end or file');

end;

h1 := b;{\* 记录分程序基地址 \*}

blkcnt := 10; { post mortem dump }

repeat

writeln( prr );

blkcnt := blkcnt-1;

if blkcnt = 0

then h1 := 0;

h2 := s[h1+4].i;{\* 记录被调用分程序在符号表中的位置 \*}

if h1 <> 0

then writeln( prr, '',tab[h2].name, 'called at', s[h1+1].i:5);

h2 := btab[tab[h2].ref].last;{\* 记录分程序最后一个标识符在符号表中的位置 \*}

while h2 <> 0 do{\* 循环指导h2为0 \*}

with tab[h2] do

begin

if obj = vvariable{\* 如果最后一个标识符种类是variable \*}

then if typ in stantyps{\* 如果类型是标准类型 \*}

then begin

write(prr,'',name,'=');{\* 输出标识符的名称 \*}

if normal{\* 如果是值形参 \*}

then h3 := h1+adr{\* h3记录地址（值形参，通过一次寻址找到的是变量值） \*}

else h3 := s[h1+adr].i;{\* 否则h3记录值（变量形参，通过一次寻址找到的仍是地址，需要再次寻址） \*}

case typ of

ints : writeln(prr,s[h3].i);{\* 若是整型，打印整数值 \*}

reals: writeln(prr,s[h3].r);{\* 若是实型，打印实数值 \*}

bools: if s[h3].b{\* 如果是布尔型，打印’true’或’false’ \*}

then writeln(prr,'true')

else writeln(prr,'false');

chars: writeln(prr,chr(s[h3].i mod 64 )){\* 如果是字符型，打印字符 \*}

end

end;

h2 := link{\* 获得同一分程序中上一个标识符的位置 \*}

end;

h1 := s[h1+3].i{\* 重新设置h1为DL块数据（本分程序在S占中将要占据的数据区长度减一） \*}

until h1 < 0

end;

writeln(prr);

writeln(prr,ocnt,' steps');{\* 输出steps数量 \*}

end; { interpret }

procedure setup; {\* 建立保留字表（key）和特定字符表（sps） \*}

begin

key[1] := 'and ';{\* key：array[1..nkw] of alfa \*}

key[2] := 'array ';

key[3] := 'begin ';

key[4] := 'case ';

key[5] := 'const ';

key[6] := 'div ';

key[7] := 'do ';

key[8] := 'downto ';

key[9] := 'else ';

key[10] := 'end ';

key[11] := 'for ';

key[12] := 'function ';

key[13] := 'if ';

key[14] := 'mod ';

key[15] := 'not ';

key[16] := 'of ';

key[17] := 'or ';

key[18] := 'procedure ';

key[19] := 'program ';

key[20] := 'record ';

key[21] := 'repeat ';

key[22] := 'then ';

key[23] := 'to ';

key[24] := 'type ';

key[25] := 'until ';

key[26] := 'var ';

key[27] := 'while ';

ksy[1] := andsy;{\* array[1..nkw] of symbol \*}

ksy[2] := arraysy;

ksy[3] := beginsy;

ksy[4] := casesy;

ksy[5] := constsy;

ksy[6] := idiv;

ksy[7] := dosy;

ksy[8] := downtosy;

ksy[9] := elsesy;

ksy[10] := endsy;

ksy[11] := forsy;

ksy[12] := funcsy;

ksy[13] := ifsy;

ksy[14] := imod;

ksy[15] := notsy;

ksy[16] := ofsy;

ksy[17] := orsy;

ksy[18] := procsy;

ksy[19] := programsy;

ksy[20] := recordsy;

ksy[21] := repeatsy;

ksy[22] := thensy;

ksy[23] := tosy;

ksy[24] := typesy;

ksy[25] := untilsy;

ksy[26] := varsy;

ksy[27] := whilesy;

sps['+'] := plus;{\* array[char]of symbol \*}

sps['-'] := minus;

sps['\*'] := times;

sps['/'] := rdiv;

sps['('] := lparent;

sps[')'] := rparent;

sps['='] := eql;

sps[','] := comma;

sps['['] := lbrack;

sps[']'] := rbrack;

sps[''''] := neq;

sps['!'] := andsy;

sps[';'] := semicolon;

end {\* setup \*};

procedure enterids;{\* 在符号表中登录标准的类型（基本类型）、标准函数和标准过程的名字，以及相应的信息 \*}

begin

enter(' ',vvariable,notyp,0); { sentinel }

enter('false ',konstant,bools,0);

enter('true ',konstant,bools,1);

enter('real ',typel,reals,1);

enter('char ',typel,chars,1);

enter('boolean ',typel,bools,1);

enter('integer ',typel,ints,1);

enter('abs ',funktion,reals,0);

enter('sqr ',funktion,reals,2);

enter('odd ',funktion,bools,4);

enter('chr ',funktion,chars,5);

enter('ord ',funktion,ints,6);

enter('succ ',funktion,chars,7);

enter('pred ',funktion,chars,8);

enter('round ',funktion,ints,9);

enter('trunc ',funktion,ints,10);

enter('sin ',funktion,reals,11);

enter('cos ',funktion,reals,12);

enter('exp ',funktion,reals,13);

enter('ln ',funktion,reals,14);

enter('sqrt ',funktion,reals,15);

enter('arctan ',funktion,reals,16);

enter('eof ',funktion,bools,17);

enter('eoln ',funktion,bools,18);

enter('read ',prozedure,notyp,1);

enter('readln ',prozedure,notyp,2);

enter('write ',prozedure,notyp,3);

enter('writeln ',prozedure,notyp,4);

enter(' ',prozedure,notyp,0);

end;

begin { main }

setup;{\* 初始化保留字和特定字符 \*}

constbegsys := [ plus, minus, intcon, realcon, charcon, ident ];{\* 初始化常量开始符号的集合 \*}

typebegsys := [ ident, arraysy, recordsy ];{\* 初始化类型开始符号的集合 \*}

blockbegsys := [ constsy, typesy, varsy, procsy, funcsy, beginsy ];{\* 初始化分程序开始符号的集合 \*}

facbegsys := [ intcon, realcon, charcon, ident, lparent, notsy ];{\* 初始化因子开始符号的集合 \*}

statbegsys := [ beginsy, ifsy, whilesy, repeatsy, forsy, casesy ];{\* 初始化语句开始符号的集合 \*}

stantyps := [ notyp, ints, reals, bools, chars ];{\* 初始化基本类型集合 \*}

lc := 0; {\* 初始化Pcode的索引 \*}

ll := 0; {\* 初始化当前行的长度 \*}

cc := 0; {\* 初始化字符计数器 \*}

ch := ' '; {\* 初始化当前字符 \*}

errpos := 0; {\* 初始化错误的位置 \*}

errs := [];{\* 初始化错误的集合 \*}

writeln( 'NOTE input/output for users program is console : ' );

writeln;

write( 'Source input file ?');

readln( inf );{\* readln一直读取到回车符为止，每行只读一次；读取代码文件名称 \*}

assign( psin, inf );{\* 把读到的文件名设置为psin输入 \*}

reset( psin );{\* reset：打开文件 \*}

write( 'Source listing file ?');

readln( outf );{\* 读取输出文件名称 \*}

assign( psout, outf );{\* 把读到的文件名设置为psout输出 \*}

rewrite( psout );{\* rewrite：新建文件，若文件已存在则覆盖 \*}

assign ( prd, 'con' );{\* 设置prd为con \*}

write( 'result file : ' );

readln( fprr );{\* 读取结果文件名称 \*}

assign( prr, fprr ); {\* 把读到的文件名设置为prr结果输出 \*}

reset ( prd );

rewrite( prr );

t := -1;{\* 初始化符号表的栈顶指针 \*}

a := 0;{\* 初始化数组信息向量表的栈顶指针 \*}

b := 1;{\* 初始化分程序表的栈顶指针 \*}

sx := 0;{\* 初始化字符串常量表的栈顶指针 \*}

c2 := 0;{\* 初始化实常量表的栈顶指针 \*}

display[0] := 1;{\* 初始化display，第0层索引的btab表的位置为1 \*}

iflag := false;{\* 初始化标志位 \*}

oflag := false;

skipflag := false;

prtables := false;

stackdump := false;

insymbol;{\* 读入符号，开始编译 \*}

if sy <> programsy{\* 如果读入的不是’program’，报错 \*}

then error(3)

else begin

insymbol;

if sy <> ident{\* 如果读入的不是普通标识符（程序名），报错 \*}

then error(2)

else begin

progname := id;{\* 赋值程序名 \*}

insymbol;

if sy <> lparent{\* 如果读入的不是’(’ \*}

then error(9)

else repeat

insymbol;

if sy <> ident{\* 如果读入的不是普通标识符（标准输入、标准输出标志） \*}

then error(2)

else begin

if id = 'input '

then iflag := true{\* 标准输入标识符置位，程序中存在标准输入 \*}

else if id = 'output '

then oflag := true{\* 标准输出标识符置位，程序中存在标准输出 \*}

else error(0);{\* 不是上述两个标识符就报错 \*}

insymbol

end

until sy <> comma;{\* 若有逗号，继续读，不是逗号则停止 \*}

if sy = rparent{\* 如果读到’)’，继续读 \*}

then insymbol

else error(4);

if not oflag then error(20){\* 如果没有输出就报错 \*}

end

end;

enterids;{\* 登录符号表 \*}

with btab[1] do{\* btab[1]保存主程序 \*}

begin

last := t;{\* 最后一个标识符位置为当前符号表栈顶 \*}

lastpar := 1;{\* 初始化设置，因为input和output不在符号表中 \*}

psize := 0;

vsize := 0;

end;

block( blockbegsys + statbegsys, false, 1 );{\* 调用分程序处理过程，主程序层次为1 \*}

if sy <> period{\* 主程序末尾必须是’.’，否则报错 \*}

then error(2);

emit(31); {\* halt \*}{\* 停止 \*}

if prtables{\* 列表打印标志位为true \*}

then printtables;{\* 打印所有的列表信息 \*}

if errs = []{\* 如果错误集没有记录 \*}

then interpret{\* 运行Pcode指令 \*}

else begin{\* 如果有错误，打印编译错误提示 \*}

writeln( psout );

writeln( psout, 'compiled with errors' );

writeln( psout );

errormsg;{\* 打印错误信息摘要 \*}

end;

writeln( psout );

close( psout );

close( prr )

end.