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

}

{\* 各类常量定义 \*}

const nkw = 27; { no. of key words } {\*保留字数量\*}

alng = 10; { no. of significant chars in identifiers } {\* Alfa范围 \*}

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 }

bmax = 20; { size of block-talbe }

amax = 30; { size of array-table }

c2max = 20; { size of real constant table }

csmax = 30; { max no. of cases }

cmax = 800; { size of code } {\* P代码表范围 \*}

lmax = 7; { maximum level } {\* P代码指令l域范围 \*}

smax = 600; { size of string-table }

ermax = 58; { max error no. } {\*最大错误数量\*}

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

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

nmax = 32767; { 2\*\*15-1 } {\* P代码指令y域范围 数字的范围\*}

lineleng = 132; { output line length }

linelimit = 200; {\* 行数限制 \*}

stacksize = 1450; {\* 数据栈大小 \*}

{\* 自定义各种类型 \*}

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);{\* Pascal-S所有关键字的枚举 \*}

index = -xmax..+xmax;

alfa = packed array[1..alng]of char; {\*10个字符组成的数组\*}

objecttyp = (konstant, vvariable, typel, prozedure, funktion ); {\*标识符的种类, 分别为常量、变量、类型、过程和函数\*}

types = (notyp, ints, reals, bools, chars, arrays, records ); {\* Pascal-S的所有数据类型的枚举，其中notyp一般对应的是过程名\*}

symset = set of symbol; {\*symbol的集合\*}

typset = set of types;

item = record

typ: types;

ref: index;

end;

{\* P代码指令域 \*}

order = packed record

f: -omax..+omax;

x: -lmax..+lmax;

y: -nmax..+nmax

end;

{\* 各类变量定义 \*}

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

rnum: real; { real number from insymbol }

inum: integer; { integer from insymbol }

sleng: integer; { string length }

cc: integer; { character counter } {\*字符计数变量\*}

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

ll: integer; { length of current line }{\*当前行的长度\*}

errpos: integer; {\* 出错位置 \*}

t,a,b,sx,c1,c2:integer; { indices to tables }

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 }

progname: alfa;

stantyps: typset;

constbegsys, typebegsys, blockbegsys, facbegsys, statbegsys: symset;

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

key: array[1..nkw] of alfa; {\* 保留字集合 \*}

ksy: array[1..nkw] of symbol; {\* 保留字对应的syn集合 \*}

sps: array[char]of symbol; { special symbols }

display: array[0..lmax] of integer; {\* display区 \*}

tab: array[0..tmax] of { indentifier lable } {\* 符号表，P440 \*}

packed record

name: alfa; {\*标识符名字，取前10个字符\*}

link: index;{\*指向同一个分程序中上一个标识符在tab表中的位置\*}

obj: objecttyp; {\*标识符种类\*}

typ: types; {\*数据类型\*}

ref: index;{\*当标识符为数组类型名或者数组变量名时，rel为指向该数组在atab中登录的位置指针值；当标识符为记录类型名或者记录变量名或过程名、函数名时，ref为指向该记录在分程序表btab中的指针值。其他情况下ref一律为0\*}

normal: boolean;

lev: 0..lmax; {\*该标识符所在分程序的静态层次，规定主程序的层次为1，递增\*}

adr: integer

end;

atab: array[1..amax] of { array-table } {\* 数组信息向量表，P442 \*}

packed record

inxtyp,eltyp: types;{\*下表索引类型和数组元素类型\*}

elref,low,high,elsize,size: index {\*数组元素为数组或记录时，elref为它在atab登记的位置，其余为0。数组的上下界、数组元素的大小和数组的大小\*}

end;

btab: array[1..bmax] of { block-table } {\* 分程序表，P441 \*}

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; {\* P代码表 目标代码区,存储编译生成的目标指令P442\*}

psin,psout,prr,prd:text; { default in pascal p } {\* 写入输入，输出，输出结果，文件的输出流 \*}

inf, outf, fprr: string; {\* 代码输入，代码输出，结果输出的文件路径 \*}

{\* 打印错误信息摘要 \*}

procedure errormsg;

var k : integer;

msg: array[0..ermax] of alfa; {\* 错误信息数组,每一条数组最多10个字符，最多有ermax种错误 \*}

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); {\* 将空行写入psout，psout为输出流 \*}

writeln(psout,'key words'); {\* 将'key words'写入psout并换行 \*}

k := 0;

{\* 循环处理未被处理的错误信息 \*}

while errs <> [] do {\*循环直到错误信息数组为空\*}

begin

while not( k in errs )do k := k + 1; {\* k从0一直递增,找到当前error的编号，存到k里 \*}

writeln(psout, k, ' ', msg[k] ); {\* 将错误编号及错误信息写入psout \*}

errs := errs - [k] {\* 从集合errs中删除当前错误 \*}

end { while errs }

end { errormsg } ;

{\* 出错后在跳过部分代码下面画下划线 \*}

procedure endskip;

begin { underline skipped part of input }

while errpos < cc do

begin

write( psout, '-');

errpos := errpos + 1

end;

skipflag := false {\* 已跳过代码并设置下划线，将flag设为false \*}

end { endskip };

{\* 预存一整行代码到line中，去除行末分号，每次都从line中读取字符 \*}

procedure nextch; { read next character; process line end }

begin

if cc = ll {\* 如果读到了当前行行尾 \*}

then begin

if eof( psin ) {\* 要读入的文件已经读完 \*}

then begin

writeln( psout ); {\* 输出空行 \*}

writeln( psout, 'program incomplete' ); {\* 输出'program incomplete' \*}

errormsg; {\* 输出错误信息 \*}

exit;

end;

if errpos <> 0 {\* errpos不等于0，有错误出现 \*}

then begin

if skipflag then endskip; {\* 调用endskip过程加下划线 \*}

writeln( psout );

errpos := 0

end;

write( psout, lc: 5, ' '); {\* 没有错误执行的操作，在list文件中输出当前PCODE的行号以及一个空格，不换。lc:5表示输出长度大于等于5,不足5前面用空格补齐\*}

ll := 0; {\* 将行长度和字符指针置零 \*}

cc := 0;

while not eoln( psin ) do {\* 因为在读字符之前当前行已经读完了，所以psin的指针指向的是下一行代码,于是进行循环将psin所在行的代码装入line中。循环读字符直到读到行末，能进入此循环说明之前处理了错误或进入新行 \*}

begin

ll := ll + 1; {\* 统计当前行长度 \*}

read( psin, ch ); {\* 读入一个字符 \*}

write( psout, ch ); {\* 输出该字符 \*}

line[ll] := ch {\* 将ch保存到line中,循环结束line保存到当前行末的所有字符 \*}

end;

ll := ll + 1;

readln( psin ); {\* 从psin读一行空行，让psin指向下一行代码 \*}

line[ll] := ' '; {\* 将行末置为空格 \*}

writeln( psout );

end;

cc := cc + 1; {\* 字符指针后移 \*}

ch := line[cc]; {\* 读取下一个字符，ch = 取出来的字符 \*}

end { nextch };

{\* 打印出错位置和错误编号，并将错误编号加入errs中 \*}

procedure error( n: integer ); {\* n为错误号 \*}

begin

if errpos = 0

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

if cc > errpos {\* 确认字符计数指针在当前errpos之后，避免重复报错 \*}

then begin

write( psout, ' ': cc-errpos, '^', n:2);

errpos := cc + 3;

errs := errs +[n] {\* 将错误号加入errs集合 \*}

end

end { error };

{\* 打印表格溢出信息，写入信息多于表大小时会终止程序 \*}

procedure fatal( n: integer ); {\* n为出错表类型 \*}

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

begin

writeln( psout );

errormsg;

msg[1] := 'identifier'; msg[2] := 'procedures';

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 };

{\* 读取下一个单词符号,处理注释行 \*}

procedure insymbol; {reads next symbol}

label 1,2,3; {\* 定义label，为goto做准备 \*}

var i,j,k,e: integer; {\* k为临时表示位数的值，e为科学计数法的指数 \*}

{\* 处理实数的指数部分 \*}

procedure readscale;

var s,sign: integer;

begin

nextch;

sign := 1; {\* 正负标识,1为正,-1为负。符号默认为正号 \*}

s := 0; {\* 数字默认为0 \*}

if ch = '+' {\* 如果读到'+',读下一个字符 \*}

then nextch

else if ch = '-' {\* 如果读到'-',符号设为负 \*}

then begin

nextch;

sign := -1

end;

if not(( ch >= '0' )and (ch <= '9' )) {\* 符号后面不是数字则报错 \*}

then error( 40 )

else repeat {\* 读完指数并存入s \*}

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

nextch;

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

e := s\*sign + e {\* 与adjustscale过程计算得到的e相加得到最后的e \*}

end { readscale };

{\* 根据小数位数和指数大小求出实数尾部值的大小，并附在rnum后面得到最后的实数 \*}

procedure adjustscale;

var s : integer;

d, t : real;

begin

if k + e > emax {\* 当前位数加上指数超过指数上限则报错 \*}

then error(21)

else if k + e < emin {\* 低于最小值则直接将实数设置为0 \*}

then rnum := 0

else begin

s := abs(e); {\* 将指数转为正数方便处理 \*}

t := 1.0; {\* 指数部分转换后的结果 \*}

d := 10.0; {\* 底数 \*}

repeat {\* 将实数的指数部分变为普通数字 \*}

while not odd(s) do {\* 循环处理偶次幂直到指数为奇数 \*}

begin

s := s div 2; {\* 指数除以二 \*}

d := sqr(d) {\* 把平方直接转到d上，d = d的平方 \*}

end;

s := s - 1;

t := d \* t

until s = 0;

if e >= 0 {\* 判断指数正负，决定是该除以t还是乘以t。把指数e转换成2N+1或者2N的形式, t此时为10的e次方\*}

then rnum := rnum \* t

else rnum := rnum / t

end

end { adjustscale };

{\* 编译选项 \*}

procedure options;

{\* 处理编译选项中的'+','-'号 \*}

procedure switch( var b: boolean ); {\* 处理编译可选项中的“+”,“-”号，传入参数为prtables或者stackdump \*}

begin

b := ch = '+';

if not b

then if not( ch = '-' )

then begin { print error message } {\* 当前符号不是加号也不是减号，报错，跳过终止符之前所有符号 \*}

while( ch <> '\*' ) and ( ch <> ',' ) do {\* 跳过所有不是'\*'与','的符号}

nextch;

end

else nextch {\* 当前符号是减号，继续处理下一个字符 \*}

else nextch {\* 当前符号是加号，继续处理下一个字符 \*}

end { switch };

begin { options } {\* 处理编译选项，t+表示打印各种表格，会将prtables设置为true

s+表示遇到运行出错时，进行现成卸出打印，会将stackdump特征变量置成true，

t-,s-与前两者用途相反 \*}

repeat

nextch;

if ch <> '\*' {\* 编译选项要求填在注释里，故开始之前会有星号 \*}

then begin

if ch = 't' {\*表示打印相关的操作\*}

then begin

nextch;

switch( prtables ) {\*根据符号判断是否打印表格,设置prtable的值\*}

end

else if ch = 's' {\*设置是否需要在出错时现场卸除打印\*}

then begin

nextch;

switch( stackdump )

end;

end

until ch <> ',' {\* 若有逗号说明设置了多个编译选项，继续处理这些编译选项 \*}

end { options };

begin { insymbol }

1: while( ch = ' ' ) or ( ch = chr(9) ) do {\* 跳过所有空格或者9号ascii字符，即'\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 } {\* 若首个字符为字符，开始识别单词 \*}

k := 0;

id := ' '; {\* 10个空格，长度与alng值相等\*}

repeat {\* 将id前k个字符设置为ch，并不断读取ch，直到ch不是小写字母或者数字 \*}

if k < alng

then begin

k := k + 1;

id[k] := ch

end;

nextch

until not((( ch >= 'a' ) and ( ch <= 'z' )) or (( ch >= '0') and (ch <= '9' ))); {\* 识别到的\*}

i := 1;

j := nkw; { binary search }

repeat {\* 二分查表,找到当前id在表中的位置 \*}

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 \*}

else sy := ident {\* 没有找到即当前单词不是保留字，为普通标识符 \*}

end;

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

begin { number } {\* 数字开头则当做数字识别 \*}

k := 0;

inum := 0;

sy := intcon; {\* 保留字类型为symbol类的intcon，表示整数\*}

repeat {\* 将整数部分全部读到inum中 \*}

inum := inum \* 10 + ord(ch) - ord('0'); {\*把小数点或者e前面的数读完存到inum中\*}

k := k + 1; {\*统计整数部分的位数\*}

nextch

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

if( k > kmax ) or ( inum > nmax ) {\* 整数位数超上限或数字大小超上限，报错 \*}

then begin

error(21);

inum := 0;

k := 0

end;

if ch = '.' {\* 开始读小数或数域 \*}

then begin

nextch;

if ch = '.' {\* 连续两个点，说明是数域，而非小数 \*}

then ch := ':'

else begin {\* 数字类型从整数变为实数 \*，rnum接管inum \*}

sy := realcon; {\* 从整数变为实数 \*}

rnum := inum; {\* 将整数部分存入inum \*}

e := 0; {\*0.1被认为是10的-1次方,以此类推\*}

while ( ch >= '0' ) and ( ch <= '9' ) do {\* 读取所有小数部分，转化为科学计数法 \*}

begin

e := e - 1;

rnum := 10.0 \* rnum + (ord(ch) - ord('0')); {\*把小数部分先当做整数来存\*}

nextch

end;

if e = 0 {\* 未读到小数部分，报错 \*}

then error(40);

if ch = 'e' {\* 如果小数后面有e(科学计数法),则进行下面操作处理指数部分 \*}

then readscale;

if e <> 0 then adjustscale {\* 计算最后结果 \*}

end

end

else if ch = 'e' {\* 处理科学计数法的指数部分。读到的数字是不带小数的实数 \*}

then begin

sy := realcon;

rnum := inum;

e := 0;

readscale;

if e <> 0

then adjustscale

end;

end;

':':

begin {\* 判断是赋值还是普通冒号 \*}

nextch;

if ch = '='

then begin

sy := becomes;

nextch

end

else sy := colon

end;

'<':

begin {\* 判断是<=还是<>还是< \*}

nextch;

if ch = '='

then begin

sy := leq;

nextch

end

else

if ch = '>'

then begin

sy := neq;

nextch

end

else sy := lss

end;

'>':

begin {\* 判断是>=还是> \*}

nextch;

if ch = '='

then begin

sy := geq;

nextch

end

else sy := gtr

end;

'.':

begin {\* 判断是..还是. \*}

nextch;

if ch = '.'

then begin

sy := colon;{\*..算作和冒号一类,处理array[1..max]一类\*}

nextch

end

else sy := period

end;

'''':{\*当前字符是否为单引号\*}

begin {\* 处理单引号及其之后的内容 \*}

k := 0;

2: nextch;

if ch = ''''

then begin

nextch;

if ch <> '''' {\*如果不是三个连续的单引号\*}

then goto 3 {\* 引号内字符串已经读完 \*}

end;

if sx + k = smax {\*sx为stab的索引变量\*}

then fatal(7); {\* 字符串表溢出，报错 \*}

stab[sx+k] := ch;

k := k + 1;

if cc = 1 {\* nextch进入新行使cc变为1，故cc=1代表一行已读完}

then begin { end of line }

k := 0;

end

else goto 2; {\* 这一行的字符未读完，继续读取 \*}

3: if k = 1 {\* 引号内为单个字符 \*}

then begin

sy := charcon; {\*sym类型为字符类型\*}

inum := ord( stab[sx] ) {\*inum存储该字符的ASCII值\*}

end

else if k = 0 {\* 空引号，报错（不支持字符串跨行） \*}

then begin

error(38);

sy := charcon;

inum := 0 {\*编码值为0\*}

end

else begin {\* 按字符串处理 \*}

sy := stringcon; {\*类型为字符串类型\*}

inum := sx; {\*如果是字符串则存储该字符的开始位置索引下标\*}

sleng := k;

sx := sx + k

end

end;

'(':

begin {\* 判断是普通的(还是编译选项 \*}

nextch;

if ch <> '\*'

then sy := lparent

else begin { comment }

nextch;

if ch = '$'

then options;

repeat

while ch <> '\*' do nextch;

nextch

until ch = ')';

nextch;

goto 1 {\* 编译选项处理完需要继续处理之后的字符 \*}

end

end;

'{':

begin {\* 判断是普通的{还是编译选项 \*}

nextch;

if ch = '$'{\*{$是编译选项设置\*}

then options;

while ch <> '}' do

nextch;

nextch;

goto 1

end;

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

begin {\* 处理特殊符号 \*}

sy := sps[ch];

nextch

end;

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

begin {\* 以上符号不可作为字符开头，报错，继续往下处理单词 \*}

error(24);

nextch;

goto 1

end

end { case }

end { insymbol };

{\* 把标准类型、过程、函数名登到符号表(tab)中。

x0为标识符名，x1为标识符种类，x2为标识符类型，x3为地址或大小（大小只针对类型） \*}

procedure enter(x0:alfa; x1:objecttyp; x2:types; x3:integer );

begin

t := t + 1; { enter standard identifier }

with tab[t] do

begin

name := x0;

link := t - 1;

obj := x1;

typ := x2;

ref := 0;

normal := true;

lev := 0; {\*主程序开始之前就登录了一些自带的,所以级别最高\*}

adr := x3;

end

end; { enter }

{\* 将数组下标信息录入数组表atab，

tp为数组下标类型，可为ints,bools或者chars

l,h分别为数组下上界 \*}

procedure enterarray( tp: types; l,h: integer );

begin

if l > h {\* 界限出错 \*}

then error(27);

if( abs(l) > xmax ) or ( abs(h) > xmax ) {\* 超范围下标，报错 \*}

then begin

error(27);

l := 0;

h := 0;

end;

if a = amax {\* 数组表已满，报错 \*}

then fatal(4)

else begin

a := a + 1;

with atab[a] do {\* 正常设置数组的三个域 \*}

begin

inxtyp := tp;

low := l;

high := h

end

end

end { enterarray };

{\* 将分程序信息录入分程序表btab \*}

procedure enterblock;

begin

if b = bmax {\* 分程序表满了 \*}

then fatal(2)

else begin

b := b + 1;

btab[b].last := 0; {\* 指向过程或函数最后一个符号在表中的位置,建表用 \*}

btab[b].lastpar := 0; {\* 指向过程或者函数的最后一个'参数'符号在tab中的位置,退栈用 \*}

end

end { enterblock };

{\* 录入实常量表rconst \*}

procedure enterreal( x: real );

begin

if c2 = c2max - 1

then fatal(3)

else begin

rconst[c2+1] := x;

c1 := 1; {\*循环用的局部变量\*}

while rconst[c1] <> x do

c1 := c1 + 1;

if c1 > c2

then c2 := c1 {\*如果在c2之前就有该常量,则c1<=c2,于是c2不更新,否则c2更新\*}

end

end { enterreal };

{\* emit和下面两个过程都是用来生成PCODE的，后个过程接的参数是操作数

fct为操作码 \*}

procedure emit( fct: integer ); {\*这几个emit是用来生成pcode的,fct为操作码,这里为无操作数\*}

begin

if lc = cmax {\*lc为code表的索引变量\*}

then fatal(6);

code[lc].f := fct;

lc := lc + 1

end { emit };

procedure emit1( fct, b: integer ); {\*一个操作数\*}

begin

if lc = cmax

then fatal(6);

with code[lc] do

begin

f := fct;

y := b;

end;

lc := lc + 1

end { emit1 };

procedure emit2( fct, a, b: integer ); {\*两个操作数\*}

begin

if lc = cmax then fatal(6);

with code[lc] do

begin

f := fct;

x := a;

y := b

end;

lc := lc + 1;

end { emit2 };

{\* 打印编译生成的符号表、分程序表、实常数表、PCODE}

procedure printtables;

var i: integer;

o: order;

mne: array[0..omax] of

packed array[1..5] of char;

begin

{\* 定义所有PCODE指令符 \*}{\*p-code指令\*}

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');

writeln(psout);

for i := btab[1].last to t do

with tab[i] do {\* 输出btab[1]中最后一个标识符到当前标识符之间所有标识符在tab中的信息 \*}

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' );

writeln( psout );

for i := 1 to b do

with btab[i] do {\* 输出所有分程序在btab中信息 \*}

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');

writeln( psout );

for i := 1 to a do

with atab[i] do {\* 输出所有数组在atab中信息 \*}

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:');

writeln( psout );

for i := 0 to lc-1 do

begin {\* 输出所有PCODE \*}

write( psout, i:5 );

o := code[i];

write( psout, mne[o.f]:8, o.f:5 );

if o.f < 31

then if o.f < 4

then write( psout, o.x:5, o.y:5 )

else write( psout, o.y:10 )

else write( psout, ' ' );

writeln( psout, ',' )

end;

writeln( psout );

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

{\*btab第一项已填入内容，故程序从btab[2]开始 \*}

end { printtables };

{\* 处理分程序 \*}

procedure block( fsys: symset; isfun: boolean; level: integer );

type conrec = record {\* 该记录可以根据tp的类型决定另一个域是i还是r \*}

case tp: types of

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

reals :( r:real )

end;

var dx : integer ; { data allocation index }

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

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

x : integer ;

{\* 跳过出错过程块.跳过错误的代码段,直到取出来的符号等于给定的符号集\*}

procedure skip( fsys:symset; n:integer);

begin

error(n);

skipflag := true;

while not ( sy in fsys ) do

insymbol;

if skipflag then endskip

end { skip };

{\* 检查当前sym是否合法,若不合法,打印出错标志并进行跳读 \*}

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

begin

if not( sy in s1 )

then skip( s1 + s2, n )

end { test };

{\* 检查分号是否合法 \*}

procedure testsemicolon;

begin

if sy = semicolon

then insymbol

else begin

error(14);

if sy in [comma, colon]

then insymbol

end;

test( [ident] + blockbegsys, fsys, 6 )

end { testsemicolon };

{\* 在分程序中将标识符id填入tab,k为标识符种类 \*}

procedure enter( id: alfa; k:objecttyp );

var j,l : integer;

begin

if t = tmax {\* tab已满，报错 \*}

then fatal(1)

else begin

tab[0].name := id;

j := btab[display[level]].last; {\* 得到对应level的分程序的最后一个标识符位置 \*}

l := j;

while tab[j].name <> id do {\* 从分程序尾部一直向前遍历，看是否存在与id重名的标识符 \*}

j := tab[j].link;

if j <> 0 {\* 有重名则报错。已经在符号表中有记载了,重复定义 \*}

then error(1)

else begin {\* 不重名则正常入栈 。在tab中登记信息\*}

t := t + 1;

with tab[t] do

begin

name := id;

link := l;

obj := k;

typ := notyp;

ref := 0;

lev := level;

adr := 0;

normal := false { initial value }

end;

btab[display[level]].last := t{\*更新当前分程序层最后一个标识符在tab表中的位置\*}

end

end

end { enter };

{\* 查找分程序中标识符id在符号表中位置并返回 \*}

function loc( id: alfa ):integer;

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

begin

i := level; {\*i为当前分程序层\*}

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

repeat

j := btab[display[i]].last;

while tab[j].name <> id do

j := tab[j].link;

i := i - 1;

until ( i < 0 ) or ( j <> 0 );{\*当前层没有则往前再找一层,直到找到(j<>0)或者没有(i<0)\*}

if j = 0

then error(0);{\*没找到,报错\*}

loc := j

end { loc } ;

{\* 将变量加入到tab，若sy不是标识符则报错 \*}

procedure entervariable;

begin

if sy = ident

then begin

enter( id, vvariable );

insymbol

end

else error(2)

end { entervariable };

{\* 处理分程序中常量，由c返回常量的类型与值 \*}

procedure constant( fsys: symset; var c: conrec );

var x, sign : integer;

begin

c.tp := notyp;

c.i := 0;

test( constbegsys, fsys, 50 );

if sy in constbegsys {\* 如果sy是常量开始的符号,才往下继续分析 \*}

then begin

if sy = charcon

then begin {\* sy是字符常量 \*}

c.tp := chars;

c.i := inum; {\* inum存储该字符的ascii码值 \*}

insymbol

end

else begin

sign := 1; {\* sy不是字符常量,默认符号为正 \*}

if sy in [plus, minus] {\* sy是正负号 \*}

then begin

if sy = minus {\*是负号\*}

then sign := -1;

insymbol

end;

if sy = ident {\* sy是标识符常量 \*}

then begin

x := loc(id); {\* 找到id在表中位置 \*}

if x <> 0

then

if tab[x].obj <> konstant {\* id对应符号种类不是常量，报错 \*}

then error(25)

else begin {\* 得到的tab[x]为标识符常量，对c进行相应处理 \*}

c.tp := tab[x].typ; {\* 得到类型 \*}

if c.tp = reals {\* 根据类型是整数还是实数进行处理 \*}

then c.r := sign\*rconst[tab[x].adr]

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

end;

insymbol

end

else if sy = intcon {\* sy是整数常量 \*}

then begin

c.tp := ints;

c.i := sign\*inum; {\* 在i域中存入带符号的整数值 \*}

insymbol

end

else if sy = realcon {\* sy是实数常量 \*}

then begin

c.tp := reals;

c.r := sign\*rnum; {\* 在r域中存入带符号的整数值 \*}

insymbol

end

else skip(fsys,50) {\* sy不是任何类型常量，报错并跳过部分代码 \*}

end;

test(fsys,[],6)

end

end { constant };

{\* 处理类型说明,返回当前关键词的类型tp,在符号表中的位置rf,以及需要占用存储空间的大小sz \*}

procedure typ( fsys: symset; var tp: types; var rf,sz:integer );

var eltp : types;

elrf, x : integer;

elsz, offset, t0, t1 : integer;

{\* 数组类型的处理比较特殊，做单独处理.登录数组类型到atab并返回数组的登录位置和数组大小 \*}

procedure arraytyp( var aref, arsz: integer );

var eltp : types; {\* 数组元素类型 \*}

low, high : conrec; {\* 上下界 \*}

elrf, elsz: integer; {\* 记录ref和size方便返回 \*}

begin

constant( [colon, rbrack, rparent, ofsy] + fsys, low ); {\* 获得数组编号下界 \*}

if low.tp = reals {\* 下标类型不是整数，报错并切换为整数，数值为0}

then begin

error(27);

low.tp := ints;

low.i := 0

end;

if sy = colon {\* 下一个符号是..或者:都可继续执行 \*}

then insymbol

else error(13);

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

if high.tp <> low.tp {\* 上下界需保持类型一致，否则报错，并将上界大小调为与下界一致 \*}

then begin

error(27);

high.i := low.i

end;

enterarray( low.tp, low.i, high.i ); {\* 将数组下标信息录入atab \*}

aref := a; {\* 将数组在atab中的位置存在aref中 \*}

if sy = comma

then begin {\* 若读到逗号，说明需要建立多维数组 \*}

insymbol;

eltp := arrays; {\* 数组元素类型为arrays \*}

arraytyp( elrf, elsz ) {\* 递归处理数组内的数组 \*}

end

else begin {\* 不是逗号的情况（右方括号或非法） \*}

if sy = rbrack {\* 读到右方括号，说明数组下标部分声明完毕 \*}

then insymbol

else begin {\* 非法 \*}

error(12);

if sy = rparent {\* 若为右大括号则容错 \*}

then insymbol

end;

if sy = ofsy {\* 读到of关键字则继续，否则报错 \*}

then insymbol

else error(8);

typ( fsys, eltp, elrf, elsz ) {\* 处理数组元素的类型 \*}

end;

with atab[aref] do {\* 处理完多维数组或数组元素信息则将这些信息存入atab[aref]中}

begin

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

size := arsz;

eltyp := eltp;

elref := elrf;

elsize := elsz

end

end { arraytyp };

begin { typ }

tp := notyp;

rf := 0;

sz := 0;

test( typebegsys, fsys, 10 ); {\* 检查当前符号是否为类型声明的开始符 \*}

if sy in typebegsys {\* 如果是类型声明的开始符 \*}

then begin

if sy = ident {\* 当前符号为标识符 \*}

then begin

x := loc(id); {\* 查找id在符号表中的位置 \*}

if x <> 0

then with tab[x] do

if obj <> typel {\* 符号表中标识符种类不是类型标识符(typel)（非用户定义或编译器自带），报错 \*}

then error(29)

else begin {\* 正常获取符号类型 \*}

tp := typ;

rf := ref;

sz := adr;{\*获得其在运行栈中分配的存储空间\*}

if tp = notyp {\* 未定义类型，报错 \*}

then error(30)

end;

insymbol

end

else if sy = arraysy {\* 当前符号为array关键字 \*}

then begin

insymbol;

if sy = lbrack {\* 当前符号为[，则处理下一符号}

then insymbol

else begin {\* 否则报错并容错( \*}

error(11);

if sy = lparent

then insymbol

end;

tp := arrays; {\*类型为数组类型\*}

arraytyp(rf,sz) {\*登录数组类型\*}

end

else begin { records } {\* 不是标识符也不是数组类型，则只可能为记录 \*}

insymbol;

enterblock; {\* 记录被看做一个分程序，故需在btab中记录它的信息 \*}

tp := records;

rf := b; {\* rf指向btab中记录的位置 \*}

if level = lmax

then fatal(5);{\* 当前嵌套层次已经是最大层次了,即不能产生更深的嵌套，报错并终止程序 \*}

level := level + 1;

display[level] := b; {\* 建立分层次索引 \*}

offset := 0;

while not ( sy in fsys - [semicolon,comma,ident]+ [endsy] ) do

{\* end之前都是记录类型变量内的变量声明 \*}

begin { field section } {\* 处理记录内部成员变量 \*}

if sy = ident

then begin {\* 当前符号为标识符 \*}

t0 := t; {\* 将当前tab指针存入t0 \*}

entervariable; {\* 变量表 \*}

while sy = comma do {\* 同类型变量同时申明，

通过逗号分隔,未遇到冒号则继续读入并入表 \*}

begin

insymbol;

entervariable

end;

if sy = colon

then insymbol{\* 遇到了冒号,说明这类的变量声明结束了,冒号后面跟变量的类型 \*}

else error(5);

t1 := t; {\* 将当前tab指针存入t1 \*}

typ( fsys + [semicolon, endsy, comma,ident], eltp, elrf, elsz );

{\* 递归调用typ来处理记录类型的成员变量,确定各成员的类型,ref和adr \*}

while t0 < t1 do {\* 填写t0到t1中信息缺失的部分,

由于t0~t1都是同一类型的变量,因此size大小是相同 \*}

begin

t0 := t0 + 1;

with tab[t0] do {\* 用获取到的成员变量信息补充表项内容 \*}

begin

typ := eltp;{\*eltp来自上面的typ递归调用\*}

ref := elrf; {\* 此处ref为记录在btab中的位置 \*}

normal := true;{\*所有normal的初值都为false\*}

adr := offset; {\* 变量地址位移 \*}

offset := offset + elsz {\* 下一变量真实位置 \*}

end

end

end; { sy = ident }

if sy <> endsy

then begin {\* 当前符号不是end \*}

if sy = semicolon

then insymbol {\* 若为分号，则正常读取后续符号 \*}

else begin {\* 否则报错并容错，容逗号 \*}

error(14);

if sy = comma

then insymbol

end;

test( [ident,endsy, semicolon],fsys,6 )

{\* 检测当前符号是否合法。开启下一行处理时检验当前符号是否合法\*}

end

end; { field section }

btab[rf].vsize := offset; {\* vsize为记录所需存储单元数目 \*}

sz := offset;

btab[rf].psize := 0; {\* 记录存储单元不需要psize。该程序块的参数占用空间为0,因为record类型并不是真正的过程,没有参数 \*}

insymbol;

level := level - 1{\*record处理结束后level降一层\*}

end; { record }

test( fsys, [],6 ) {\*检查当前sym是否合法\*}

end;

end { typ };

{\* 处理过程或函数说明中的形参，将形参信息填入符号表 \*}

procedure parameterlist; { formal parameter list }

var tp : types;

valpar : boolean; {\* value parameter \*}

rf, sz, x, t0 : integer;

begin

insymbol;

tp := notyp;

rf := 0;{\*初始化符号表的位置\*}

sz := 0; {\*初始化元素的大小\*}

test( [ident, varsy], fsys+[rparent], 7 );

while sy in [ident, varsy] do {\* 处理所有是标识符或var关键字的形参 \*}

begin

if sy <> varsy {\* 不是var（指针）参数，将valpar设置为true \*}

then valpar := true {\*值形参\*}

else begin {\* 是指针参数，将valpar设置为false \*}

insymbol;

valpar := false

end;

t0 := t; {\* 存第一个参数在tab中地址到t0 \*}

entervariable;

while sy = comma do {\* 循环给多个同类型参数入表 \*}

begin

insymbol;

entervariable;

end;

{\* 以下代码与typ处理记录同类成员变量的代码近似，只做不同部分的分析 \*}

if sy = colon{\*等于冒号则开始处理类型\*}

then begin

insymbol;

if sy <> ident {\* 非标识符一定不是数据类型，报错 \*}

then error(2)

else begin

x := loc(id);

insymbol;

if x <> 0

then with tab[x] do

if obj <> typel {\*是标识符中的类型\*}

then error(29)

else begin

tp := typ;

rf := ref;

if valpar {\* 值形参。针对指针参数与值参数对sz做不同处理 \*}

then sz := adr {\*获得当前形参在符号表的位置\*}

else sz := 1 {\*不是值形参则将其置为1(在这篇代码中,变量形参的大小都设置为了1)\*}

end;

end;

test( [semicolon, rparent], [comma,ident]+fsys, 14 )

end

else error(5); {\* 不是冒号则报错（不支持数据类型的自动解释） \*}

while t0 < t do

begin

t0 := t0 + 1;

with tab[t0] do

begin

typ := tp;

ref := rf; {\* ref = 0 \*}

adr := dx; {\* adr = 运行栈中存储分配单元的相对地址dx \*}

lev := level;

normal := valpar;

dx := dx + sz{\*block中的变量存储索引更新\*}

end

end;

if sy <> rparent {\* 不是右括号代表还有参数或出错 \*}

then begin

if sy = semicolon

then insymbol

else begin

error(14);

if sy = comma

then insymbol

end;

test( [ident, varsy],[rparent]+fsys,6)

end

end { while };

if sy = rparent {\* 参数声明结束后应当用右括号结尾 \*}

then begin

insymbol;

test( [semicolon, colon],fsys,6 )

end

else error(4)

end { parameterlist };

{\* 处理常量声明，将常量信息填入tab \*}

procedure constdec;

var c : conrec;

begin

insymbol;

test([ident], blockbegsys, 2 ); {\* 判断sy是否为标识符 \*}

while sy = ident do {\* 若为标识符，则处理所有同类型申明 \*}

begin

enter(id, konstant); {\* id作为常数种类入符号表 \*}

insymbol;

if sy = eql {\* 常量赋值用等号 \*}

then insymbol

else begin {\* 否则出错，容错:= \*}

error(16);

if sy = becomes {\*：=\*}

then insymbol

end;

constant([semicolon,comma,ident]+fsys,c); {\* c获得常量的数据类型和值 \*}

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

tab[t].ref := 0; {\* 常量ref域为0 \*}

if c.tp = reals {\* c为实数 \*}

then begin

enterreal(c.r); {\* c入实常量表rconst \*}

tab[t].adr := c1; {\* tab[t]的adr域填入c在rconst中位置c1 \*}

end

else tab[t].adr := c.i; {\* c为整数，则填入数值到tab[t]的adr \*}

testsemicolon

end

end { constdec };

{\* 处理类型自定义，将自定义的类型信息填入tab，与constdec过程几乎一样 \*}

procedure typedeclaration;

var tp: types;

rf, sz, t1 : integer;

begin

insymbol;

test([ident], blockbegsys,2 );{\*检查获取到的是不是标识符\*}

while sy = ident do

begin

enter(id, typel);

t1 := t;

insymbol;

if sy = eql {\*赋值符号识别与容错\*}

then insymbol

else begin

error(16);

if sy = becomes

then insymbol

end;

typ( [semicolon,comma,ident]+fsys, tp,rf,sz ); {\* 通过调用typ过程获取类型填表所需域 \*}

with tab[t1] do {\*将typ的返回值填表\*}

begin

typ := tp;

ref := rf;

adr := sz {\* 类型的adr存的是类型所需大小 \*}

end;

testsemicolon

end

end { typedeclaration };

{\* 处理普通变量申明，将变量填入tab，与parameterlist的值形参处理过程几乎一样 \*}

{\*处理变量类型,并将变量名及其相应信息填入符号表\*}

procedure variabledeclaration;

var tp : types;

t0, t1, rf, sz : integer;

begin

insymbol;

while sy = ident do {\*是标识符\*}

begin

t0 := t;

entervariable; {\*将变量名登录到符号表中\*}

while sy = comma do {\*多个变量名逗号分隔\*}

begin

insymbol;

entervariable;

end;

if sy = colon {\*冒号\*}

then insymbol

else error(5);

t1 := t;

typ([semicolon,comma,ident]+fsys, tp,rf,sz );{\*获得类型地址和大小\*}

while t0 < t1 do {\*对一行的变量循环填表\*}

begin

t0 := t0 + 1;

with tab[t0] do

begin

typ := tp;

ref := rf;

lev := level;

adr := dx;

normal := true;

dx := dx + sz

end

end;

testsemicolon

end

end { variabledeclaration };

{\* 处理过程与函数声明，将函数名或过程名填入tab \*}

procedure procdeclaration;

var isfun : boolean; {\*是否是function\*}

begin

isfun := sy = funcsy; {\* sy是function就是方法，否则是过程 \*}

insymbol;

if sy <> ident {\* sy不是标识符则报错，用十个空格代替 \*}

then begin

error(2);

id :=' '

end;

if isfun {\* 函数或过程入tab \*}

then enter(id,funktion)

else enter(id,prozedure);

tab[t].normal := true;

insymbol;

block([semicolon]+fsys, isfun, level+1 ); {\* 递归调用block处理分程序 \*}

if sy = semicolon {\* 读到分号才算过程正常结束，否则报错 \*}

then insymbol

else error(14);

emit(32+ord(isfun)) {exit} {\* 生成退出分程序的PCODE \*}

end { proceduredeclaration };

{\* 分析处理各种语句 \*}

procedure statement( fsys:symset );

var i : integer;

{\* 处理表达式子程序，提前声明供selector调用，避免蛋生鸡问题。由参数(x)返回求值结果的类型 \*}

procedure expression(fsys:symset; var x:item); forward;

{\* 处理结构变量v（数组下标或记录的成员变量）的使用 \*}

procedure selector(fsys:symset; var v:item);

var x : item;

a,j : integer;

begin { sy in [lparent, lbrack, period] } {\* 只要sy是(或者[或者.，就一直处理 \*}

repeat

if sy = period {\* sy是.，后续内容作为成员变量处理 \*}

then begin

insymbol; { field selector }

if sy <> ident {\* 域的类型必为标识符，否则报错 \*}

then error(2)

else begin

if v.typ <> records {\* v不是records类型，报错 \*}

then error(31)

else begin { search field identifier }

j := btab[v.ref].last; {\* 获得该记录在tab中最后一个标识符的位置 \*}

tab[0].name := id; {\* 暂存id \*}

while tab[j].name <> id do {\* 在tab中，

从j的位置（记录最后一项）向前查找id \*}

j := tab[j].link;

if j = 0 {\* 未找到该标识符，说明该域在记录中不存在，报错 \*}

then error(0);

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

v.ref := tab[j].ref; {\*记录其在btab中的指针值\*}

a := tab[j].adr; {\*记录该变量对record变量起始地址的相对位移\*}

if a <> 0 {\*如果相对位移不为0则生成一条指令来计算此位移\*}

then emit1(9,a) {\* 输出PCODE:INT a，将a放在栈顶 \*}

end;

insymbol

end

end

else begin { array selector } {\*处理数组下标。sy是[或(或其他，后续内容作为数组下标处理或报错 \*}

if sy <> lbrack {\* 只认[作为取数组操作符，对{做隐式容错处理 \*}

then error(11);

repeat {\* 循环读取方括号中所有内容， 此处循环是为了满足处理多维数组的需要\*}

insymbol;

expression( fsys+[comma,rbrack],x); {\* 递归调用expression处理下标 \*}{\*处理表达式,例如array[6+i]\*}

if v.typ <> arrays

then error(28)

else begin

a := v.ref; {\*从v中获得该数组在atab中的位置\*}

if atab[a].inxtyp <> x.typ {\*如果传入的下标类型和atab中记录的数组下标类型不符则报错\*}

then error(26)

else if atab[a].elsize = 1 {\* 输出PCODE去取下标变量地址，如果是变量形参(指针)\*}

then emit1(20,a) {\*IDX,取下标变量地址,长度为1\*}

else emit1(21,a); {\*IXX,取下标变量地址\*}

v.typ := atab[a].eltyp;

v.ref := atab[a].elref

end

until sy <> comma; {\*直到没有遇到逗号终止循环\*}

if sy = rbrack {\* 遇到右括号，正常结束，否则报错 \*}

then insymbol

else begin

error(12);

if sy = rparent {\*对右括号容错处理\*}

then insymbol

end

end

until not( sy in[lbrack, lparent, period]);

test( fsys,[],6)

end { selector };

{\* 处理非标准的过程或函数调用

i表示需要调用的过程或函数名在符号表中的位置 \*}

procedure call( fsys: symset; i:integer );

var x : item; {\*代表传进来的参数结构体\*}

lastp,cp,k : integer;

begin

emit1(18,i); { mark stack } {\* 生成标记栈指令,传入被调用过程或函数在tab表中的位置,建立新的内务信息区 \*}

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

cp := i; {\*记录当前被调用函数或者过程在符号表中的位置\*} {\*cp 到 lastp 之间是tab表中形参列表,若是在循环中cp > lastp则表示实参与形参个数不一致\*}

if sy = lparent {\*如果识别到左括号则开始进行被调用过程的参数处理\*}

then begin { actual parameter list }

repeat

insymbol;

if cp >= lastp {\* 如果当前符号的位置小于最后一个符号的位置，说明还有参数没有处理,反之报错 \*}

then error(39)

else begin {\* 开始处理参数 \*}

cp := cp + 1;

if tab[cp].normal {\* 如果normal的值为真，即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);

{\* 调用expression处理参数 \*}

if x.typ = tab[cp].typ {\*表达式的类型和分程序要求输入的类型一致\*}

then begin

if x.ref <> tab[cp].ref {\* btab中地址不同，报错 \*}

then error(36)

else if x.typ = arrays {\* x为数组类型，生成装入块指令，将实参表达式的值或地址放到预留的参数单元中 \*}

then emit1(22,atab[x.ref].size)

else if x.typ = records {\* x为记录类型，同样生成装入块指令完成操作，只是细节有所不同 \*}

then emit1(22,btab[x.ref].vsize)

end

else if ( x.typ = ints ) and ( tab[cp].typ = reals )

{\* 如果表达式的类型是整型，但是分程序要求输入的是实数型参数 \*}

then emit1(26,0) {\* 生成强制转换指令 \*}

else if x.typ <> notyp

then error(36);

end

else begin { variable parameter }

if sy <> ident {\*如果传进来的不是标识符则报错\*}

then error(2)

else begin

k := loc(id);{\*找到其在tab中的位置\*}

insymbol;

if k <> 0

then begin

if tab[k].obj <> vvariable {\* 获取到的形参种类不是变量，报错 \*}

then error(37);

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

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

if tab[k].normal {\* tab[k]是值形参，将变量地址装入栈顶 \*}

then emit2(0,tab[k].lev,tab[k].adr)

else emit2(1,tab[k].lev,tab[k].adr);

{\* 是变量形参，将变量的值装入栈顶 \*}

if sy in [lbrack, lparent, period]{\*后面跟的可以是左中括号(数组下标),左括号(容错)或.(对应记录)。若有则调用分析子结构的过程来处理 \*}

then selector(fsys+[comma,colon,rparent],x);

if ( x.typ <> tab[cp].typ ) or ( x.ref <> tab[cp].ref )

then error(36)

end

end

end {variable parameter }

end;

test( [comma, rparent],fsys,6)

until sy <> comma; {\*循环直到处理完逗号\*}

if sy = rparent {\* 以括号结束 \*}

then insymbol

else error(4)

end;

if cp < lastp {\* 实参数量不够，报错 \*}

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

emit1(19,btab[tab[i].ref].psize-1 ); {\* 生成CAL指令，正式开始过程或函数调用 \*}

if tab[i].lev < level

then emit2(3,tab[i].lev, level ) {\* 生成DIS指令，更新display区 \*}

end { call };

{\* 处理整型或实数型两个操作数运算时的类型转换。(a + b = c, 根据a 和 b 判断c的类型)\*}

function resulttype( a, b : types) :types;

begin

if ( a > reals ) or ( b > reals ) {\* 操作数不是整型也不是实数型，报错，返回notyp \*}

then begin

error(33);

resulttype := notyp

end

else if ( a = notyp ) or ( b = notyp ) {\* 操作数都是notyp，返回notyp \*}

then resulttype := notyp

else if a = ints {\* 操作数a是整型 \*}

then if b = ints {\* b是整型则不转换，b是实型则转实型 \*}

then resulttype := ints

else begin

resulttype := reals;

emit1(26,1) {\* 生成指令FLT进行类型转化 \*}

end

else begin {\* a不是整型，返回结果必是实型 \*}

resulttype := reals;

if b = ints {\* b是整型，生成FLT进行类型转化 \*}

then emit1(26,0)

end

end { resulttype } ;

procedure expression( fsys: symset; var x: item ); {\*处理表达式,由x返回求值结果的类型\*}

var y : item;

op : symbol;

procedure simpleexpression( fsys: symset; var x: item ); {\*处理简单表达式(+、-、or),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,传入标准函数的编号n,执行不同的操作\*}

var ts : typset; {\*类型集合\*}

begin { standard function no. n }

if sy = lparent {\*如果当前符号不是左括号则报错\*}

then insymbol

else error(9);

if n < 17 {\*如果标准函数编号小于17\*}

then begin

expression( fsys+[rparent], x ); {\*递归调用处理表达式的过程来处理参数\*}

case n of

{ abs, sqr } 0,2: begin

ts := [ints, reals];

tab[i].typ := x.typ;

if x.typ = reals {\*如果参数类型为实数则对应的函数标号+1\*}

then n := n + 1

end;

{ odd, chr } 4,5: ts := [ints]; {\* 判定奇偶与ascii码转字符串只涉及整型 \*}

{ odr } 6: ts := [ints,bools,chars]; {\* 6号操作涉及整型,布尔型或者字符型 \*}

{ succ,pred } 7,8 : begin {\* 得到前、后一个元素涉及整型、布尔型或者字符型 \*}

ts := [ints, bools,chars];

tab[i].typ := x.typ

end;

{ round,trunc } 9,10,11,12,13,14,15,16: {\* 其他17以前数学运算涉及ints与reals \*}

{ sin,cos,... } begin

ts := [ints,reals];

if x.typ = ints {\* 若为整型，则转为实型并生成PCODE \*}

then emit1(26,0)

end;

end; { case }

if x.typ in ts

then emit1(8,n) {\* x满足涉及类型，生成FCT n，表示生成n号标准函数 \*}

else if x.typ <> notyp

then error(48); {\*该标准函数的参数类型不正确\*}

end

else begin { n in [17,18] } {\*17、18号，eof 和 eoln 函数\*}

if sy <> ident {\*不是标识符\*}

then error(2)

else if id <> 'input ' {\* 只有‘input ’合法。报0号错误,未知id\*}

then error(0)

else insymbol;

emit1(8,n); {\* 生成对应标准函数 \*}

end;

x.typ := tab[i].typ;

if sy = rparent {\* 右括号 标准函数调用结束\*}

then insymbol

else error(4)

end { standfct } ;

begin { factor } {\* 因子分析程序开始 \*}

x.typ := notyp;

x.ref := 0;

test( facbegsys, fsys,58 );

while sy in facbegsys do {\* 从所有合法因子开始符号开始循环处理因子 \*}

begin

if sy = ident {\* sy是普通标识符 \*}

then begin

i := loc(id);

insymbol;

with tab[i] do

case obj of

konstant: begin {\* 处理常量标识符并生成PCODE \*}

x.typ := typ;

x.ref := 0;

if x.typ = reals {\* x是否实常数 \*}

then emit1(25,adr) {\* LDR adr：将实数装入数据栈,

注意实数常量的adr对应着其在rconst实常量表中的位置 \*}

else emit1(24,adr) {\* LDC adr：装入字面常量，

如果是整型直接存入栈顶即可 \*}

end;

vvariable:begin {\* 处理变量标识符并生成PCODE \*}

x.typ := typ;

x.ref := ref;

if sy in [lbrack, lparent,period] {\* x是数组或记录 \*} {\*如果该标识符变量后面紧跟'('、']'、'.'则说明该变量存在子结构\*}

then begin

if normal

then f := 0 {\*值形参标准名或其他变量\*}

else f := 1; {\*变量形参\*}

emit2(f,lev,adr); {\*变量形参装载地址而值形参装载值\*}

selector(fsys,x); {\*处理子结构得到真正的变量\*}

if x.typ in stantyps

then emit(34)

end

else begin {\* x是普通变量 \*}

if x.typ in stantyps

then if normal

then f := 1 {\*标准类型的值形参进行取值操作\*}

else f := 2 {\*标准类型的变量形参进行间接取值操作\*}

else if normal

then f := 0 {\*非标准类型的值形参取地址\*}

else f := 1; {\*非标准类型的变量形参取值 ？？？？\*}

emit2(f,lev,adr)

end

end;

typel,prozedure: error(44); {\* 因子不能为类型名或过程。表达式中不能出现的类型或者过程标识符\*}

funktion: begin {\*如果是函数符号 处理函数并生成PCODE \*}

x.typ := typ;

if lev <> 0 {\* 通过lev是否为0判断处理的是

标准函数还是非标准函数。层次为0则是标准函数,层次不为零则不是标准函数\*}

then call(fsys,i)

else standfct(adr)

end

end { case,with }

end

else if sy in [ charcon,intcon,realcon ] {\* sy是常量 \*}

then begin

if sy = realcon {\* 处理实常量 \*}

then begin

x.typ := reals;

enterreal(rnum); {\*将该元素放入实常量表\*}

emit1(25,c1) {\* LDR c1，

将实常量表中第c1个(也就是刚刚放进去的)元素放入栈顶 \*}

end

else begin {\*整数和字符型常量\*}

if sy = charcon

then x.typ := chars

else x.typ := ints;

emit1(24,inum) {\* LDC inum：装入字面常量inum \*}

end;

x.ref := 0;

insymbol

end

else if sy = lparent {\* sy是左括号，处理括号中表达式 \*}

then begin

insymbol;

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

if sy = rparent

then insymbol

else error(4)

end

else if sy = notsy {\* sy是逻辑非关键字 \*}

then begin

insymbol;

factor(fsys,x); {\* 递归调用factor \*}

if x.typ = bools {\* x的type为布尔型，生成NOT:逻辑非指令 \*}

then emit(35)

else if x.typ <> notyp {\* x是其他type，报错 \*}

then error(32)

end;

test(fsys,facbegsys,6)

end { while }

end { factor };

begin { term } {\*项\*}

factor( fsys + [times,rdiv,idiv,imod,andsy],x); {\* 利用factor分析[]中因子 \*}

while sy in [times,rdiv,idiv,imod,andsy] do{\*因子和因子之间用\* / DIV MOD AND\*}

begin

op := sy;

insymbol;

factor(fsys+[times,rdiv,idiv,imod,andsy],y ); {\* 处理二元操作的第二个操作数 \*}

if op = times {\* 操作符为乘号 \*}

then begin

x.typ := resulttype(x.typ, y.typ);

case x.typ of

notyp: ;

ints : emit(57); {\* MUL：整形乘 \*}

reals: emit(60); {\* MUR：实型乘 \*}

end

end

else if op = rdiv {\* 操作符为实型除 \*}

then begin

if x.typ = ints {\* x强制转换为实型 \*}

then begin

emit1(26,1);

x.typ := reals;

end;

if y.typ = ints {\* y强制转换为实型 \*}

then begin

emit1(26,0);

y.typ := reals;

end;

if (x.typ = reals) and (y.typ = reals)

then emit(61) {\* DIR: 实型除 \*}

else begin {\* x或y为notyp或出错 \*}

if( x.typ <> notyp ) and (y.typ <> notyp)

then error(33);

x.typ := notyp

end

end

else if op = andsy {\* 操作符为与操作 \*}

then begin

if( x.typ = bools )and(y.typ = bools) {\* x与y必须为bools \*}

then emit(56) {\* AND：逻辑与 \*}

else begin {\* x或y为notyp或出错 \*}

if( x.typ <> notyp ) and (y.typ <> notyp)

then error(32);

x.typ := notyp

end

end

else begin { op in [idiv,imod] } {\* 操作符为整型除或整型取模 \*}

if (x.typ = ints) and (y.typ = ints)

then if op = idiv

then emit(58) {\* DIV：整数除 \*}

else emit(59) {\* MOD: 取模 \*}

else begin {\* x或y为notyp或出错 \*}

if ( x.typ <> notyp ) and (y.typ <> notyp)

then error(34);

x.typ := notyp

end

end

end { while }

end { term };

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

if sy in [plus,minus] {\* sy是正负号 \*}

then begin

op := sy;

insymbol;

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

if x.typ > reals

then error(33)

else if op = minus {\* 减号取相反数，MUS：求负 \*}

then emit(36)

end

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

while sy in [plus,minus,orsy] do {\* 循环处理出现的+，-，or \*}

begin

op := sy;

insymbol;

term(fsys+[plus,minus,orsy],y);

if op = orsy {\* 处理or \*}

then begin

if ( x.typ = bools )and(y.typ = bools)

then emit(51) {\* ORR：逻辑或 \*}

else begin

if( x.typ <> notyp) and (y.typ <> notyp)

then error(32);

x.typ := notyp

end

end

else begin {\* 处理算术运算符+- \*}

x.typ := resulttype(x.typ,y.typ); {\* 得到类型转换后的值 \*}

case x.typ of

notyp: ;

ints: if op = plus

then emit(52) {\* ADD \*}

else emit(53); {\* SUB \*}

reals:if op = plus

then emit(54) {\* ADR \*}

else emit(55) {\* SUR \*}

end { case }

end

end { while }

end { simpleexpression };

begin { expression }

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

if sy in [ eql,neq,lss,leq,gtr,geq] {\* sy是数值比较符 \*}

then begin

op := sy;

insymbol;

simpleexpression(fsys,y);

if(x.typ in [notyp,ints,bools,chars]) and (x.typ = y.typ)

then case op of {\* 操作数类型一致且不为实型才开始比较，

以下所有PCODE都只针对整型 \*}

eql: emit(45); {\* EQL \*}

neq: emit(46); {\* NEQ \*}

lss: emit(47); {\* LSS \*}

leq: emit(48); {\* LEQ \*}

gtr: emit(49); {\* GRT \*}

geq: emit(50); {\* GEQ \*}

end

else begin {\* 操作数类型不一致则做转换 \*}

if x.typ = ints {\* 转换x为reals \*}

then begin

x.typ := reals;

emit1(26,1)

end

else if y.typ = ints {\* 转换y为reals \*}

then begin

y.typ := reals;

emit1(26,0)

end;

if ( x.typ = reals)and(y.typ=reals)

then case op of {\* 针对实型数值比较运算的PCODE \*}

eql: emit(39); {\* EQR \*}

neq: emit(40); {\* NER \*}

lss: emit(41); {\* LSR \*}

leq: emit(42); {\* LER \*}

gtr: emit(43); {\* GTR \*}

geq: emit(44); {\* GER \*}

end

else error(35)

end;

x.typ := bools {\* 将返回的x值变成bools}

end

end { expression };

{\* 处理赋值语句 \*}

procedure assignment( lv, ad: integer );

var x,y: item;

f : integer;

begin { tab[i].obj in [variable,prozedure] } {\*当且仅当当前符号表的目标类型为变量或者过程型时\*}

x.typ := tab[i].typ;

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

if tab[i].normal

then f := 0

else f := 1;

emit2(f,lv,ad);

if sy in [lbrack,lparent,period] {\* 出现左括号和.，变量为数组或记录 \*}

then selector([becomes,eql]+fsys,x);

if sy = becomes {\* 读到赋值符号，之后开始赋值 \*}{\*赋值符号并容错\*}

then insymbol

else begin

error(51);

if sy = eql {\* 等号容错 \*}

then insymbol

end;

expression(fsys,y); {\* 获得赋值符号右侧值 \*}

if x.typ = y.typ {\* 左右类型相等的处理 \*}

then if x.typ in stantyps {\* x不是数组变量也不是记录变量，就是普通的变量 \*}

then emit(38) {\* STO：栈顶赋值到次栈顶 \*}

else if x.ref <> y.ref {\* x，y是记录变量但ref与y不等，说明不是同一个记录类型 \*}

then error(46)

else if x.typ = arrays {\* x是数组 \*}

then emit1(23,atab[x.ref].size) {\* CPB atab[x.ref].size：拷贝atab中的项 \*}

else emit1(23,btab[x.ref].vsize) {\* x是记录且记录类型与y一样 CPB atab[x.ref].size：拷贝btab中的项 \*}

else if(x.typ = reals )and (y.typ = ints)

then begin

emit1(26,0); {\* FLT 0 \*}{\* y类型需转换为reals \*}

emit(38) {\* STO \*} {\*赋值\*}

end

else if ( x.typ <> notyp ) and ( y.typ <> notyp )

then error(46)

end { assignment };

{\* 处理带begin end的混合语句 \*}

procedure compoundstatement;

begin

insymbol;

statement([semicolon,endsy]+fsys); {\* 处理一句语句 \*}

while sy in [semicolon]+statbegsys do {\* sy是分号或者statbegsys就循环处理混合语句 \*}

begin

if sy = semicolon

then insymbol

else error(14);

statement([semicolon,endsy]+fsys) {\* 处理一句语句 \*}

end;

if sy = endsy {\* 遇到end，说明当前层正常结束 \*}

then insymbol

else error(57)

end { compoundstatement };

{\* 处理if语句 \*}

procedure ifstatement;

var x : item;

lc1,lc2: integer;

begin

insymbol;

expression( fsys+[thensy,dosy],x); {\* 将if到then或到do之前的内容丢进expression处理 \*}

if not ( x.typ in [bools,notyp]) {\* 得到的应该是布尔值，否则报错 \*}

then error(17);

lc1 := lc;

emit(11); { jmpc } {\* JPC:若x为假，跳转到y域对应地址 \*}

if sy = thensy {\* 处理then之后内容 \*}

then insymbol

else begin {\* 没加then，报错，对do进行容错 \*}

error(52);

if sy = dosy

then insymbol

end;

statement( fsys+[elsesy]); {\* 处理到else之前ifthen的代码 \*}

if sy = elsesy {\* 处理else内代码 \*}

then begin

insymbol;

lc2 := lc;

emit(10); {\* JMP:无条件跳转到y域对应地址 \*}

code[lc1].y := lc; {\*设置if指令跳转的位置\*}

statement(fsys); {\*处理else条件下的分析语句\*}

code[lc2].y := lc {\*设置else指令跳转的位置\*}

end

else code[lc1].y := lc {\*设置if指令跳转的位置\*}

end { ifstatement };{\*设置if指令跳转的位置\*}

{\* 处理case语句 \*}

procedure casestatement;

var x : item;

i,j,k,lc1 : integer;

casetab : array[1..csmax]of {\* 限制case表个数为csmax \*}

packed record

val,lc : index

end;

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

{\* 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中，并检查标号有无重复定义 \*}

procedure caselabel;

var lab : conrec;

k : integer;

begin

constant( fsys+[comma,colon],lab );{\*处理常量(因为标签都是常量)\*}

if lab.tp <> x.typ {\*如果获得的标签类型和变量的类型不符\*}

then error(47){\*case语句中标号必须是与case子句表达式类型相同的常量\*}

else if i = csmax {\*如果可以声明的case达到了最大限度。爆栈报错并终止程序 \*}

then fatal(6)

else begin

i := i+1; {casestatement的变量i为case总数。得到新case，移动case指针 \*}

k := 0; {\* 用来检查标号是否重复定义的变量 \*}

casetab[i].val := lab.i; {\* 保存新case值（即i指针） \*}

casetab[i].lc := lc; {\* 记录新case生成代码的位置 \*}

repeat

k := k+1

until casetab[k].val = lab.i; {\* 扫一遍已经声明的label,看有没有重复声明 \*}

if k < i {\* 出现重复声明，报错 \*}

then error(1); { multiple definition }

end

end { caselabel };

{\* 处理case语句的一个分支 \*}

procedure onecase;

begin

if sy in constbegsys {\* case之后必须是常量 \*}

then begin

caselabel; {\* 获取一个标签 \*}

while sy = comma do {\* 如果有逗号说明是一个case对应多个标签的情况，则继续处理下个标签 \*}

begin

insymbol;

caselabel

end;

if sy = colon {\* 读到冒号说明label声明结束，读到其他符号，报错 \*}

then insymbol

else error(5);

statement([semicolon,endsy]+fsys); {\* 处理冒号到分号间内容 \*}

j := j+1; {\* 用来记录当前case对应exittab的位置 \*}

exittab[j] := lc; {\* 记录当前case分支结束的代码位置,即下面将要生成的跳转指令的位置 \*}

emit(10) {\* JMP: 生成一条跳转指令来结束这一case分支 \*}

end

end { onecase };

begin { casestatement }

insymbol;

i := 0;

j := 0;

expression( fsys + [ofsy,comma,colon],x ); {\* 处理到of \*}

if not( x.typ in [ints,bools,chars,notyp ]) {\* 若结果不是以上四种类型则报错 \*}

then error(23);

lc1 := lc; {\* 记录当前PCODE代码的位置指针 \*}

emit(12); {jmpx} {\* SWT:查找情况表,注意这里暂时没有给定跳转的地址 \*}

if sy = ofsy

then insymbol

else error(8);

onecase; {\* 处理一个分支 \*}

while sy = semicolon do {\* 循环处理之后所有分支 \*}

begin

insymbol;

onecase

end;

code[lc1].y := lc;

for k := 1 to i do {\* 遍历所有分支，建立情况表 \*}

begin

emit1( 13,casetab[k].val); {\* CAS：设置查找的值 \*}

emit1( 13,casetab[k].lc); {\* CAS：给出对应的跳转地址 \*}

end;

emit1(10,0); {\* JMP：说明情况表结束 \*}

for k := 1 to j do {\* 给定每个case分支退出之后的跳转地址 \*}

code[exittab[k]].y := lc; {\* 现在的lc指向情况表结束之后的位置，

将各分支的结束跳转地址指向这里 \*}

if sy = endsy {\* endsy以示结束 \*}

then insymbol

else error(57)

end { casestatement };

{\* 处理repeat语句 \*}

procedure repeatstatement;

var x : item;

lc1: integer; {\* 用来记录repeat的开始位置 \*}

begin

lc1 := lc; {\* 保存repeat当开始时的代码地址 \*}

insymbol;

statement( [semicolon,untilsy]+fsys); {\* 调用statement递归子程序来处理循环体中的语句 \*}

while sy in [semicolon]+statbegsys do

{\* 如果遇到了分号或者statement的开始符号,则说明循环体中还有语句没有处理完 \*}

begin

if sy = semicolon

then insymbol

else error(14);

statement([semicolon,untilsy]+fsys) {\* 处理循环体下一条语句 \*}

end;

if sy = untilsy {\* 遇到until，处理until内内容 \*}

then begin

insymbol;

expression(fsys,x);

if not(x.typ in [bools,notyp] ) {\* until内内容必须为bools，否则报错 \*}

then error(17);

emit1(11,lc1);

end

else error(53)

end { repeatstatement };

{\* 处理while语句 \*}

procedure whilestatement;

var x : item;

lc1,lc2 : integer;

begin

insymbol;

lc1 := lc;

expression( fsys+[dosy],x); {\* 处理do之前内容 \*}

if not( x.typ in [bools, notyp] ) {\* 内容必须为bools \*}

then error(17);

lc2 := lc;

emit(11); {\* JPC \*}

if sy = dosy {\* do之后正常处理 \*}

then insymbol

else error(54);

statement(fsys); {\* 正常处理 \*}

emit1(10,lc1); {\* JMP lc1：往回跳转 \*}

code[lc2].y := lc

end { whilestatement };

{\* 处理for语句 \*}

procedure forstatement;

var cvt : types; {\*循环变量的类型\*}

x : item;

i,f,lc1,lc2 : integer;

begin

insymbol;

if sy = ident {\* for语句开头是标识符 \*}

then begin

i := loc(id); {\* 从tabs中获得计数变量标识符 \*}

insymbol;

if i = 0 {\* 找不到标识符则计数变量，将计数变量类型默认处理为整形 \*}

then cvt := ints

else if tab[i].obj = vvariable {\* 对应的这个标识符对应符号是变量类型，正常处理 \*}

then begin

cvt := tab[i].typ;

if not tab[i].normal {\* 如果是变量形参，则报错 \*}

then error(37)

else emit2(0,tab[i].lev, tab[i].adr ); {\* 如果不是变量形参，

获取该符号的地址 \*}

if not ( cvt in [notyp, ints, bools, chars])

then error(18)

end

else begin {\* 符号的类型也不是变量，报错并将计数变量类型设置为整型 \*}

error(37); {\*应该是变量类型\*}

cvt := ints

end

end

else skip([becomes,tosy,downtosy,dosy]+fsys,2); {\* for语句开头不是标识符或什么也没有，

跳过出错内容 \*}

if sy = becomes {\* 读到赋值符号，给计数器赋初值 \*}

then begin

insymbol;

expression( [tosy, downtosy,dosy]+fsys,x); {\*for a:= 6 + 1 to 10 do ... 此处处理6+1并将结果的类型返回\*}

if x.typ <> cvt {\*如果和a 的类型不符,则报错\*}

then error(19);

end

else skip([tosy, downtosy,dosy]+fsys,51); {\* 没出现赋值符号，跳过出错内容 \*}

f := 14; {\* F1U \*}

if sy in [tosy,downtosy] {\* 读到to或downto，加步伐大小 \*}

then begin

if sy = downtosy {\* downto就 F1D：减步伐大小 \*}

then f := 16;

insymbol;

expression([dosy]+fsys,x); {\* 处理到do之前 \*}

if x.typ a<> cat

then error(19)

end

else skip([dosy]+fsys,55); {\* 没出现to或downto，跳过出错内容 \*}

lc1 := lc; {\* 记录下句F1U指令的位置 \*}

emit(f); {\* 生成f对应PCODE \*}

if sy = dosy {\* 找到do \*}

then insymbol

else error(54);

lc2 := lc; {\* 获取循环体开始代码的位置 \*}

statement(fsys); {\* 处理循环体语句 \*}

emit1(f+1,lc2); {\* 结束时生成F2U或F2D指令 \*}

code[lc1].y := lc {\* 将之前产生的F1U的跳转地址回传回去 \*}

end { forstatement };

{\* 处理标准过程（输入、输出） \*}

procedure standproc( n: integer );

var i,f : integer;

x,y : item;

begin

case n of

1,2 : begin { read }

if not iflag {\* input flag为true才开始读入 \*}

then begin

error(20); {\*程序头部未包含参数“output”或“input”\*}

iflag := true

end;

if sy = lparent {\* 左括号后为读入内容 \*}

then begin

repeat

insymbol;

if sy <> ident {\* 读入参数应该为标识符 \*}

then error(2)

else begin

i := loc(id);

insymbol;

if i <> 0

then if tab[i].obj <> vvariable

then error(37)

else begin

x.typ := tab[i].typ;

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

if tab[i].normal {\*指令处理过程中其操作的都是地址,装载值则直接将地址赋值,加载地址则根据地址再去找值\*}

then f := 0 {\*对值形参加载地址,根据地址索引到S中的值,所以为加载地址\*}

else f := 1; {\*对变量形参加载值,即 a=adr,所以加载的地址直接赋值就好了,即指令为加载值\*}

emit2(f,tab[i].lev,tab[i].adr);

if sy in [lbrack,lparent,period]

then selector( fsys+[comma,rparent],x);

if x.typ in [ints,reals,chars,notyp]

then emit1(27,ord(x.typ)) {\* 若n = 1，读入1个字符 \*}

else error(41) {\*read或write的参数类型不正确\*}

end

end;

test([comma,rparent],fsys,6);

until sy <> comma;

if sy = rparent {\* 右括号表示结束 \*}

then insymbol

else error(4)

end;

if n = 2

then emit(62) {\* 读入一行 \*}

end;

3,4 : begin { write }

if sy = lparent {\* 左括号后为写入内容 \*}

then begin

repeat {\* 循环读字符串，可用逗号分割 \*}

insymbol;

if sy = stringcon {\*如果是字符串类型\*}

then begin

emit1(24,sleng); {\* LDC sleng：装入sleng字面常量 \*}

emit1(28,inum); {\* WRS inum：写字符 \*}

insymbol

end

else begin {\* 写入内容带表达式 \*}

expression(fsys+[comma,colon,rparent],x); {\* 处理到逗号、冒号或右括号 \*}

if not( x.typ in stantyps ) {\* x不可为数组或记录 \*}

then error(41);

if sy = colon {\*如果出现冒号继续处理,每一个输出参数后面有冒号的话,说明有格式化操作\*}

then begin

insymbol;

expression( fsys+[comma,colon,rparent],y);

if y.typ <> ints {\*一个冒号后面代表输出几个字符\*}

then error(43); {\*write语句的域宽应该是整型\*}

if sy = colon

then begin

if x.typ <> reals {\*该表达式应该是实型\*}

then error(42);

insymbol;

expression(fsys+[comma,rparent],y);

if y.typ <> ints

then error(43);

emit(37) {\* 写实数，给定位宽 \*}

end

else emit1(30,ord(x.typ)) {\* 写，给定位宽 \*}

end

else emit1(29,ord(x.typ)) {\* 写，隐含位宽 \*}

end

until sy <> comma;

if sy = rparent {\* 右括号代表结束 \*}

then insymbol

else error(4)

end;

if n = 4

then emit(63) {\* 写一行 \*}

end; { write }

end { case };

end { standproc } ;

{\*处理分析语句\*}

begin { statement }

if sy in statbegsys+[ident]

then case sy of {\* 根据不同sy类型决定使用哪种statement子过程处理 \*}

ident : begin

i := loc(id);

insymbol;

if i <> 0

then case tab[i].obj of

konstant,typel : error(45); {\*无法对常量进行赋值操作\*}

vvariable: assignment( tab[i].lev,tab[i].adr); {\*对变量则处理赋值语句\*}

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

then call(fsys,i)

else standproc(tab[i].adr);

funktion: if tab[i].ref = display[level]

then assignment(tab[i].lev+1,0)

else error(45) {\*对函数标识符也进行赋值操作？？？？\*}

end { case }

end;

beginsy : compoundstatement;

ifsy : ifstatement;

casesy : casestatement;

whilesy : whilestatement;

repeatsy: repeatstatement;

forsy : forstatement;

end; { case }

test( fsys, [],14);

end { statement };

{\*block块刚一进入开头不是function就是procdure,所以根据一个isfun传参便可确定\*}

begin { block }

dx := 5; {\* dx是变量存储分配的索引,预设为5是为了给内务信息区留出空间 \*}

prt := t; {\* 获取当前tab的指针 \*}

if level > lmax {\*如果当前子程序的层次已经超过了允许的最大层次\*}

then fatal(5);

test([lparent,colon,semicolon],fsys,14);

enterblock; {\* 记录block信息 \*}

prb := b; {\* 获取当前btab的指针 \*} {\*获取分程序表的位置\*}

display[level] := b; {\*设置分程序索引表的相应值\*}

tab[prt].typ := notyp; {\*设置tab表的值\*}

tab[prt].ref := prb; {\*tab的ref记录在btab中的位置\*}

if ( sy = lparent ) and ( level > 1 )

then parameterlist; {\* sy为括号且level比1大，处理参数 \*}{\*处理形式参数表\*}

btab[prb].lastpar := t; {\* 以下设置两个btab域 \*} {\*过程或函数的最后一个参数在tab中的位置\*}

btab[prb].psize := dx;{\*参数及该分程序在运行栈S中的内务信息区所占的存储单元数。dx在paramaterlist过程里面也进行了改变,在5的基础上增加了形参的总大小\*}

if isfun {\* 分程序为函数 \*}

then if sy = colon {\* 冒号后面是函数返回值的类型 \*}

then begin

insymbol; { function type }

if sy = ident {\* 标识符才可能是正常类型 \*}

then begin

x := loc(id);

insymbol;

if x <> 0

then if tab[x].typ in stantyps

then tab[prt].typ := tab[x].typ

else error(15) {\*函数结果必须是integer,real,boolean或char类型\*}

end

else skip( [semicolon]+fsys,2 ) {\*略过出错部分\*}

end

else error(5);

if sy = semicolon {\* 最后分号代表分程序声明完毕，且声明结束必须有分号 \*}

then insymbol

else error(14);

repeat {\* 声明分程序的各类常变量 \*} {\*循环处理变量和函数/过程声明部分\*}

if sy = constsy

then constdec;

if sy = typesy

then typedeclaration;

if sy = varsy

then variabledeclaration;

btab[prb].vsize := dx; {\*记录局部变量、参数和内务信息区在S栈中所占的存储单元总数\*}

while sy in [procsy,funcsy] do {\*如果一个分程序块里面还有小的分程序，对照430页图\*}

procdeclaration;

test([beginsy],blockbegsys+statbegsys,56)

until sy in statbegsys;

tab[prt].adr := lc; {\*对于过程名或函数名，应填入它们相应目标代码的入口地址\*}

insymbol;

statement([semicolon,endsy]+fsys); {\*处理代码块\*}

while sy in [semicolon]+statbegsys do

begin {\* 开始处理分程序中语句 \*}

if sy = semicolon

then insymbol

else error(14);

statement([semicolon,endsy]+fsys);

end;

if sy = endsy {\* end标志分程序结束 \*}

then insymbol

else error(57);

test( fsys+[period],[],6 )

end { block };

{\* P代码解释执行过程 \*}

procedure interpret; {\*P-code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,计组pc值\*}

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

b : integer; { base index } {\*基址地址\*}

h1,h2,h3: integer;

lncnt,ocnt,blkcnt,chrcnt: integer; { counters } {\*各种技术变量\*}

ps : ( run,fin,caschk,divchk,inxchk,stkchk,linchk,lngchk,redchk ); {\* 各种错误信息标志 \*}

fld: array [1..4] of integer; { default field widths }

display : array[0..lmax] of integer;

s : array[1..stacksize] of { blockmark: } {\*运行栈\*}

Record {\*由一个个record组成\*}

case cn : types of { s[b+0] = fct result }

ints : (i: integer ); { s[b+1] = return adr }

reals :(r: real ); { s[b+2] = static link }

bools :(b: boolean ); { s[b+3] = dynamic link }

chars :(c: char ) { s[b+4] = table index }

end;

{\* 程序运行时，卸出打印现场剖析信息（display、t、b及运行栈S的内容） \*}

procedure dump;

var p,h3 : integer;

begin

h3 := tab[h2].lev; {\*该标识符所在程序的静态层次。主程序为1，递增\*}

writeln(psout);

writeln(psout);

writeln(psout,' calling ', tab[h2].name );{\*标识符名字，取前10字符\*}

writeln(psout,' level ',h3:4);

writeln(psout,' start of code ',pc:4);

writeln(psout);

writeln(psout);

writeln(psout,' contents of display ');

writeln(psout);

for p := h3 downto 0 do

writeln(psout,p:4,display[p]:6); {\*每个分程序都可以根据它的层次从display表中查到他的btab表中相应登记项的位置\*}

writeln(psout);

writeln(psout);

writeln(psout,' top of stack ',t:4,' frame base ':14,b:4);{\*栈顶指针，基址地址\*}

writeln(psout);

writeln(psout);

writeln(psout,' stack contents ':20);

writeln(psout);

for p := t downto 1 do {\*从栈顶指针开始向下\*}

writeln( psout, p:14, s[p].i:8); {\*运行栈信息\*}

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

end; {dump }

{\* 对值操作类PCODE所对应的操作 \*}

{\*以下为PCODE 0-63条指令所对应的操作\*}

procedure inter0;

begin

case ir.f of

0 : begin { load addrss } {\* 取地址操作,LDA \*}

t := t + 1; {\*栈顶指针上移\*}

if t > stacksize {\*如果超过了栈的大小上限\*}

then ps := stkchk {\*将ps设置为stkchk记录错误类型\*}

else s[t].i := display[ir.x]+ir.y {\*完成取值, 实际地址 = level起始地址+位移地址,放到栈顶\*}

end;

1 : begin { load value } {\* 取值操作,LOD \*}

t := t + 1;

if t > stacksize

then ps := stkchk

else s[t] := s[display[ir.x]+ir.y] {\*传入的是地址,完成之后将值放到栈顶\*}

end;

2 : begin { load indirect } {\* 间接取值,LDI \*}

t := t + 1;

if t > stacksize

then ps := stkchk

else s[t] := s[s[display[ir.x]+ir.y].i] {\*间接取值\*}

end;

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

h1 := ir.y;

h2 := ir.x;

h3 := b;

repeat

display[h1] := h3;

h1 := h1-1;

h3 := s[h3+2].i

until h1 = h2

end;

8 : case ir.y of {\* 标准函数，其中s[t].i代表整数，s[t].r代表实数 \*}

0 : s[t].i := abs(s[t].i);

1 : s[t].r := abs(s[t].r);

2 : s[t].i := sqr(s[t].i);

3 : s[t].r := sqr(s[t].r);

4 : s[t].b := odd(s[t].i);

5 : s[t].c := chr(s[t].i); {\* ascii码转化为字符char \*}

6 : s[t].i := ord(s[t].c); {\* 字符x转化为ascii码 \*}

7 : s[t].c := succ(s[t].c); {\* 求字符x的后继字符,比如'a'的后继是'b' \*}

8 : s[t].c := pred(s[t].c); {\* 求字符x的前导字符 \*}

9 : s[t].i := round(s[t].r); {\* 求x的四舍五入 \*}

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

11 : s[t].r := sin(s[t].r); {\* 求正弦sin(x),注意x为实数弧度，下同 \*}

12 : s[t].r := cos(s[t].r);

13 : s[t].r := exp(s[t].r);

14 : s[t].r := ln(s[t].r);

15 : s[t].r := sqrt(s[t].r);

16 : s[t].r := arcTan(s[t].r);

17 : begin

t := t+1;

if t > stacksize

then ps := stkchk

else s[t].b := eof(prd)

end;

18 : begin

t := t+1;

if t > stacksize

then ps := stkchk

else s[t].b := eoln(prd)

end;

end;

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

end { case ir.y }

end; { inter0 }

{\* 对跳转类PCODE所对应的操作 \*}

procedure inter1;

var h3, h4: integer;

begin

case ir.f of

10 : pc := ir.y ; { jump } {\* 调到第y条指令代码,JMP \*} {\*无条件跳转到y\*}

11 : begin { conditional jump } {\* 条件跳转语句,JPC \*} {\*如栈顶内容为假，转移到y\*}

if not s[t].b

then pc := ir.y;

t := t - 1 {\*退栈....因为判断结果只是临时存储,之后的判断会有新的结果\*}

end;

12 : begin { switch } {\* 转移到y的地址,查找情况表,情况表由一系列f为13的指令构成 \*}

h1 := s[t].i; {\*记录栈顶值\*}

t := t-1;{\*退栈\*}

h2 := ir.y; {\*记录要跳转的地址\*}

h3 := 0; {\*标记是否找到case分支要跳转的地方，=1则为出错或找到\*}

repeat

if code[h2].f <> 13 {\*跳转的不是情况表\*}

then begin

h3 := 1;

ps := caschk

end

else if code[h2].y = h1 {\*跳到情况表里面了，且对应的分支label值等于case表达式\*}

then begin

h3 := 1;

pc := code[h2+1].y {\*在case处理部分每个label都对应两个cas指令，第二个指令的参数是跳转地址\*}

end

else h2 := h2 + 2 {\*+2跳过该case生成的两条CAS指令，进入下一次判断\*}

until h3 <> 0

end;

14 : begin { for1up } {\* 增量步长for循环的初始判断,F1U \*}{\*增量for(to)的初始判断,其中s[t-2].i存储的是计数变量的地址,s[t-1].i存储的是计数变量值(刚进来时是初值),s[s[t-2].i].i存储的是计数变量的终值\*}

h1 := s[t-1].i; {\*第一次判断将计数变量的初值赋值给h1\*}

if h1 <= s[t].i

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

else begin

t := t - 3; {\*退去三格\*}

pc := ir.y {\*跳出循环,这里的y由后面的语句回传得到\*}

end

15 : begin { for2up } {\* 增量步长的结束判断,F2U \*}

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

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

if h1 <= s[t].i

then begin

s[h2].i := h1; {\*不断增长s[t-1].i的值\*}

pc := ir.y {\*跳到statement继续循环\*}

end

else t := t-3;

end;

16 : begin { for1down } {\* 减量步长for循环的入口判断,F1U \*}

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 } {\* 减量步长的在入口判断,F2U \*}

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,MKS \*}

h1 := btab[tab[ir.y].ref].vsize; {\*获得当前过程所需要的栈空间的大小\*}

if t+h1 > stacksize

then ps := stkchk

else begin

t := t+5; {\*预留内务信息区\*}

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

s[t].i := ir.y {\*栈顶存放被调用过程在tab表中的位置\*}

end

end;

19 : begin { call } {\* 过程或函数调用过程,CAL \*}

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

h2 := s[h1+4].i; { h2 points to tab } {\*h2指向过程名在tab表中的位置\*}

h3 := tab[h2].lev; {\*h3记录当前过程或函数的层次\*}

display[h3+1] := h1; {\*新建一个层次,并将该层次基址指向当前层次基址\*}

h4 := s[h1+3].i+h1; {\*动态链DL的值\*}

s[h1+1].i := pc; {\*返回地址\*}

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，限制长度1,IDX \*}

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

h2 := atab[h1].low; {\*atab中的上下界是1...n存储的是1 和 n\*}

h3 := s[t].i; {\*h3存放的是arry[3]中的3\*}

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) {\*数组存储基址加上偏移\*}

end

end;

21 : begin { index } {\* 取下标变量地址,IXX \*}

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,LDB \*}

h1 := s[t].i; {\*获取栈顶的值\*}

t := t-1;

h2 := ir.y+t; {\*获取块的分配空间 大小为ir.y\*}

if h2 > stacksize

then ps := stkchk

else while t < h2 do {\*循环将h1指向的块的值装入栈顶\*}

begin

t := t+1;

s[t] := s[h1];

h1 := h1+1

end

end;

23 : begin { copy block } {\*拷贝块\*}

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表示类型）\*}

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; {\*退栈。b为基址地址\*}

pc := s[b+1].i; {\*PC指向RA\*}

b := s[b+3].i {\*获得返回后的base基址,s[b+3]指向DL\*}

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 {\*给定域宽写实数\*}

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 }

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

t := t-2

end;

39 : begin

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

t := t-1;

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

end;

41 : begin

t := t-1;

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

end;

42 : begin

t := t-1;

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

end;

43 : begin

t := t-1;

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

end;

44 : begin

t := t-1;

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

end;

45 : begin

t := t-1;

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

end;

46 : begin

t := t-1;

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

end;

47 : begin

t := t-1;

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

end;

48 : begin

t := t-1;

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

end;

49 : begin

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

t := t-1;

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

end;

51 : begin

t := t-1;

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

end;

52 : begin

t := t-1;

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

end;

53 : begin

t := t-1;

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

end;

54 : begin

t := t-1;

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

end;

55 : begin

t := t-1;

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

end;

56 : begin

t := t-1;

s[t].b := s[t].b and s[t+1].b {\*逻辑与\*}

end;

57 : begin {\*整型乘\*}

t := t-1;

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

end;

58 : begin

t := t-1;

if s[t+1].i = 0 {\*除数与取模都不能为0\*}

then ps := divchk

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

end;

59 : begin

t := t-1;

if s[t+1].i = 0

then ps := divchk {\*除数与取模都不能为0\*}

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

t := t-1;

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

end;

61 : begin

t := t-1;

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

end;

62 : if eof(prd) {\*readln读完一行\*}

then ps := redchk

else readln;

63 : begin {\*writeln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; {\*所有分程序的初始入口在tab中的地址\*}

display[0] := 0;

display[1] := 0;

t := btab[2].vsize-1;

b := 0;

pc := tab[s[4].i].adr; {\*目标代码的入口地址\*}

lncnt := 0;

ocnt := 0;

chrcnt := 0;

ps := run;

fld[1] := 10;

fld[2] := 22;

fld[3] := 10;

fld[4] := 1;

repeat {\* 不断读PCODE直到结束或者报错 \*}

ir := code[pc];

pc := pc+1;

ocnt := ocnt+1;

case ir.f div 10 of

0 : inter0;

1 : inter1;

2 : inter2;

3 : inter3;

4 : inter4;

5 : inter5;

6 : inter6;

end; { case }

until ps <> run;

if ps <> fin {\* 表示编译出错，下面打印错误信息 \*}

then begin

writeln(prr);

write(prr, ' halt at', pc :5, ' because of ');

case ps of {\* 根据不同的错误信息来进行报错 \*}

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;

blkcnt := 10; { post mortem dump }

repeat

writeln( prr ); {\*写入result文件\*}

blkcnt := blkcnt-1;

if blkcnt = 0

then h1 := 0;

h2 := s[h1+4].i; {\*获取过程或者函数名在tab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; {\*获取分程序中说明的最后一个标识符在tab表中的位置\*}

while h2 <> 0 do {\*循环处理分程序中的标识符\*}

with tab[h2] do

begin

if obj = vvariable

then if typ in stantyps

then begin

write(prr,'',name,'=');

if normal

then h3 := h1+adr

else h3 := s[h1+adr].i;

case typ of

ints : writeln(prr,s[h3].i);

reals: writeln(prr,s[h3].r);

bools: if s[h3].b

then writeln(prr,'true')

else writeln(prr,'false');

chars: writeln(prr,chr(s[h3].i mod 64 ))

end

end;

h2 := link

end;

h1 := s[h1+3].i {\*指向分程序的DL,即上一层\*}

until h1 < 0

end;

writeln(prr);

writeln(prr,ocnt,' steps');

end; { interpret }

{\* 设置各类保留字与其对应符号 \*}

procedure setup;

begin

{\* 定义一系列关键字 \*}

key[1] := 'and ';

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 ';

{\* 定义关键字对应符号 \*}

ksy[1] := andsy;

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;

{\* 定义特殊字符对应的符号 \*}

sps['+'] := plus;

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 };

{\* 设置各类标准标识符与标准函数，存入符号表中 \*}

{\*在符号表中登记标准类型、函数和过程的名字以及它们的相应信息\*}

procedure enterids;

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; {\* 初始化各类标识符、保留字 \*}

constbegsys := [ plus, minus, intcon, realcon, charcon, ident ]; {\* 常量的开始符号集合 \*}

typebegsys := [ ident, arraysy, recordsy ]; {\* 类型声明的开始符号集合 \*}

blockbegsys := [ constsy, typesy, varsy, procsy, funcsy, beginsy ]; {\* 分程序的开始符号集合 \*}

facbegsys := [ intcon, realcon, charcon, ident, lparent, notsy ]; {\* 因子的开始符号集合 \*}

statbegsys := [ beginsy, ifsy, whilesy, repeatsy, forsy, casesy ]; { \* 语句开始的符号集合 \*}

stantyps := [ notyp, ints, reals, bools, chars ]; {\* 标准类型集合（排除数组与记录类型） \*}

lc := 0; {\* 初始化pc \*}

ll := 0; {\* 初始化当前行的长度 \*}

cc := 0; {\* 初始化当前行位置指针 \*}

ch := ' '; {\* 初始化当前字符 \*}

errpos := 0; {\* 初始化当前错误指针 \*}

errs := []; {\* 初始化当前错误集合 \*}

writeln( 'NOTE input/output for users program is console : ' );

writeln;

write( 'Source input file ?'); {\* 代码输入文件 \*}

readln( inf );

assign( psin, inf );

reset( psin );

write( 'Source listing file ?'); {\* 代码输出文件 \*}

readln( outf );

assign( psout, outf );

rewrite( psout );

assign ( prd, 'con' );

write( 'result file : ' ); {\* 结果输出文件 \*}

readln( fprr );

assign( prr, fprr );

reset ( prd );

rewrite( prr );

t := -1; {\* 初始化tab栈顶指针 \*}

a := 0; {\* 初始化atab栈顶指针 \*}

b := 1; {\* 初始化btab栈顶指针 \*}

sx := 0; {\* 初始化stab栈顶指针 \*}

c2 := 0; {\* 初始化rconst栈顶指针 \*}

display[0] := 1; {\* 初始化display \*}

iflag := false;

oflag := false;

skipflag := false;

prtables := false;

stackdump := false;

insymbol; {\* 开始编译 \*}

if sy <> programsy {\* 要求第一个符号是program关键字，否则报错}

then error(3)

else begin

insymbol;

if sy <> ident {\* 第二个标识符应该是程序名，否则报错 \*}

then error(2)

else begin

progname := id;

insymbol;

if sy <> lparent {\* 左括号代表进入程序头处理 \*}

then error(9)

else repeat {\* 以逗号为分割处理参数 \*}

insymbol;

if sy <> ident

then error(2)

else begin

if id = 'input '

then iflag := true

else if id = 'output '

then oflag := true

else error(0);

insymbol

end

until sy <> comma;

if sy = rparent {\* 右括号代表程序头处理完毕 \*}

then insymbol

else error(4);

if not oflag then error(20)

end

end;

enterids; {\* 加载各类标准标识符与标准函数到符号表中 \*}

with btab[1] do {\* 初始化btab[1] \*}

begin

last := t;

lastpar := 1;

psize := 0;

vsize := 0;

end;

block( blockbegsys + statbegsys, false, 1 ); {\* 开始编译主程序 \*}

if sy <> period {\* 主程序后面得跟句号，否则报错}

then error(2);

emit(31); { halt }

if prtables {\* 程序有编译参数，需要打印各类表 \*}

then printtables;

if errs = []

then interpret {\* 没有编译错误，开始启动翻译程序 \*}

else begin

writeln( psout );

writeln( psout, 'compiled with errors' );

writeln( psout );

errormsg;

end;

writeln( psout );

close( psout );

close( prr )

end.