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

}

{

Comments modified by B.W.Yang

Department of Computer Science&Engineering BUAA,07. Nov.2018

}

const nkw = 27; { no. of key words } {系统中保留字个数}

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

llng = 121; { input line length } {输入的长度}

emax = 322; { max exponent of real numbers } {实数的最大指数}

emin = -292; { min exponent } {实数的最小指数}

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

tmax = 100; { size of table } {符号表table的大小}

bmax = 20; { size of block-talbe } {分程序表btable的大小}

amax = 30; { size of array-table } {数组信息向量表atable的大小}

c2max = 20; { size of real constant table } {实数常量表的大小}

csmax = 30; { max no. of cases } {case最大情况}

cmax = 800; { size of code } {代码最大行数}

lmax = 7; { maximum level } {最大层数}

smax = 600; { size of string-table } {字符串常量表的大小}

ermax = 58; { max error no. } {最大错误种类}

omax = 63; { highest order code } {P-Code最大的指令数，共64个}

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

nmax = 32767; { 2\*\*15-1 } {第N个变量的最大范围}

lineleng = 132; { output line length }

linelimit = 200;

stacksize = 1450; {栈大小}

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);

{symbol作为枚举类型，包括整形常量，实型常量，字符常量，字符串常量，not标识，+号标识，-号标识，乘法，整数除法，实数除法，整数取模，

与，或，等于，不等于，大于，大于等于，小于，小于等于，

lparent左小括号，rparent右小括号，lbrack左中括号，rbrack右中括号，逗号，分号，句号，

冒号，becomes？，常量标识，自定义标识，变量标识，函数标识，

过程标识，数组标识，记录标识，程序标识，ident？，

开始标识，if标识，case标识，repeat标识，while标识，for标识，

end标识，else标识，until标识，of标识，do标识，to标识，downto标识，then标识}

index = -xmax..+xmax; {与上述xmax相呼应，？}

alfa = packed array[1..alng]of char; {alfa是压缩的字符数组，长度为alng，取前10个关键字符}

objecttyp = (konstant, vvariable, typel, prozedure, funktion ); {标识符的种类，有常量，变量，类型，过程和函数}

types = (notyp, ints, reals, bools, chars, arrays, records ); {标识符的类型，有notyp无类型，整型，实型，布尔型，字符型，数组型和记录型}

symset = set of symbol; {symset是枚举类型symnol的集合}

typset = set of types; {typset是枚举类型types的集合}

item = record {item是记录，包括两个枚举类型，typ和ref}

typ: types;

ref: index;

end;

order = packed record {order是压缩的记录，包括三个枚举类型，f表示P-code指令数，x表示相差层数，y是变量的次序位置}

f: -omax..+omax;

x: -lmax..+lmax;

y: -nmax..+nmax

end;

{下面是全局变量}

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 }

lc: integer; { program location counter }

ll: integer; { length of current line }

errpos: integer; {error的位置}

t,a,b,sx,c1,c2:integer; { indices to tables } {各种各样的Table的指针}

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

sy: symbol; { last symbol read by insymbol }

errs: set of 0..ermax;

id: alfa; { identifier from insymbol } {使用alfa枚举类型限制数量}

progname: alfa; {使用alfa枚举类型限制数量}

stantyps: typset;

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

line: array[1..llng] of char; {输入的字符数组，长度为llng}

key: array[1..nkw] of alfa; {nkw个保留字的关键字符}

ksy: array[1..nkw] of symbol; {nkw个保留字的符号标识}

sps: array[char]of symbol; { special symbols } {用char作为数组索引？}

display: array[0..lmax] of integer; {分程序索引表，大小由最大深度lmax决定，表项的内容就是指向分程序表btable的指针}

tab: array[0..tmax] of { indentifier lable } {！符号表，最大数量由tmax决定，每一项为一个record}

packed record

name: alfa; {name是标识符的名字，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，每个分程序在tab中登录的第一个标识符的link=0}

obj: objecttyp; {标识符的种类，为objecttyp的枚举变量}

typ: types; {标识符的类型，为types的枚举变量}

ref: index; {当标识符为数组类型名或数组变量名时，ref为指向该数组在数组信息向量表atable中的指针；

当标识符为记录类型名或记录变量名时，ref指向该记录在分程序表btable中的指针；

当标识符为过程名或函数名时，ref为指向该proc或func在btable中的指针；

其他情况，ref一律为0}

normal: boolean; {当标识符为变量形参时填入False，值形参或其他变量填入true；

标准名填true；

常量和类型名不填}

lev: 0..lmax; {表示该标识符所在分程序的静态层次，主程序层次为1，签到分程序的层数逐次+1}

adr: integer {对于变量，包括形参名，应填入该变量在运行栈S中分配存储单元的相对地址；

对于记录域名，应填入相对该记录变量起始地址的位移量；

对于过程名或函数名，应填入他们相对应代码的入口地址；

对于整数、布尔或字符常量名，应分别填入它们相应的整数值、布尔值或ASCII码；

对于实常量名，则在adr域中填入在rconst表中登录的位置；

对于类型名，应该填入该类型数据所需存储单元的大小}

end;

atab: array[1..amax] of { array-table } {数组向量信息表}

packed record

inxtyp,eltyp: types; {inxtyp是索引类型，可以使用ints、bools或chars；

eltyp是数组元素类型，可以为任意类型}

elref,low,high,elsize,size: index {elref当数组元素为数组时，elref为其在atable中登录项的位置；

当数组元素为记录时，elref为其在atable中登录的位置；

其他情况=0；

low表示数组的下界；

high表示数组的上界；

elsize为元素的大小；

size为数组的大小}

end;

btab: array[1..bmax] of { block-table } {每当编译扫描一个新的分程序时，要在btab中加入相关信息}

packed record

last, lastpar, psize, vsize: index {last始终指向该分程序中说明的最后一个标识符在table表中的位置；

lastpar指向该过程或函数的最后一个参数在table表中的位置；

psize参数及该分程序在运行栈S中的内务信息区的大小；

vsize局部变量、参数及内无信息区在S中的大小}

end;

stab: packed array[0..smax] of char; { string table } {字符串长量表，每次遇到字符串就登录到stable中，把起始位置记录在inum变量中，

把长度记录在slen变量中}

rconst: array[1..c2max] of real; {实常量表，所有遇到的字面常量都要登录在表中，实常量名的adr域不是值，而是在rconst表中的位置指针}

code: array[0..cmax] of order; {编译生成的目标指令都要放在code表中，就是目标代码区}

psin,psout,prr,prd:text; { default in pascal p } {？text是文件类型？}

inf, outf, fprr: string;

procedure errormsg;

var k : integer;

msg: array[0..ermax] of alfa; {msg为最多存放ermax个错误信息的数组，数组元素为10位标识字符串}

begin

msg[0] := 'undef id '; msg[1] := 'multi def '; {一共ermax=58种错误类型，将简化的字符串信息放入msg中}

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); {向psout-test文件中写入空行}

writeln(psout,'key words'); {向psout中写入"key words"}

k := 0;

while errs <> [] do {当errs不为空时循环}

begin

while not( k in errs )do k := k + 1; {如果第K个错误已经被删除了，那么K=K+1}

writeln(psout, k, ' ', msg[k] ); {向psout中写入“K K的错误提示”}

errs := errs - [k] {在errs中删除元素K}

end { while errs }

end { errormsg } ;

procedure endskip; {结束skip，将被skipped的输入部分下面写入-，即表示被划掉}

begin { underline skipped part of input }

while errpos < cc do {当错误发生的位置<字符计数的位置时循环}

begin

write( psout, '-'); {向psout中写入-}

errpos := errpos + 1 {errpos后移1位}

end;

skipflag := false {循环结束，将跳过flag置false}

end { endskip };

procedure nextch; { read next character; process line end } {读取下一个字符，存入ch}

begin

if cc = ll {ll为当前行的大小，cc为当前字符的位置，如果相等说明该行处理完毕}

then begin

if eof( psin ) {如果psin文件读完了}

then begin

writeln( psout ); {写空行}

writeln( psout, 'program incomplete' ); {提示程序不完全}

errormsg; {调用错误处理过程}

exit; {退出}

end;

if errpos <> 0 {如果错误位置不为0，说明可能有错误}

then begin

if skipflag then endskip; {如果skipflag为true，执行endskip过程，跳过错误代码}

writeln( psout ); {写空行}

errpos := 0 {将errpos重置为0}

end;

write( psout, lc: 5, ' '); {向psout中写入当前文件的行数}

ll := 0; {将行长度=0}

cc := 0; {将行内指针位置=0}

while not eoln( psin ) do {如果psin文件没有读完}

begin

ll := ll + 1; {行长度+1}

read( psin, ch ); {读一个字符到ch}

write( psout, ch ); {将ch写入psout}

line[ll] := ch {将ch写入字符数组}

end;

ll := ll + 1;

readln( psin ); {读取下一行}

line[ll] := ' '; {行末置为space}

writeln( psout );

end;

cc := cc + 1; {如果这一行没处理完，那么cc+1}

ch := line[cc]; {ch取出line下一个字符}

end { nextch };

procedure error( n: integer ); {错误处理，需要错误的编号，即第多少个error}

begin

if errpos = 0 {如果在初始位置，输出\*\*\*\*}

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

if cc > errpos {如果有被跳过的字符}

then begin

write( psout, ' ': cc-errpos, '^', n:2); {写入出错代码的长度和编号}

errpos := cc + 3; {将errpos+3个以跳过^和n}

errs := errs +[n] {把n加入errs集合中}

end

end { error };

procedure fatal( n: integer ); {打印表格溢出信息，如果table不够用时，引发致命错误}

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

begin

writeln( psout );

errormsg;

msg[1] := 'identifier'; msg[2] := 'procedures'; {7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} {读取下一个单词符号，处理注释行}

label 1,2,3; {？不太懂}

var i,j,k,e: integer;

procedure readscale; {处理实数的整数部分}

var s,sign: integer;

begin

nextch;

sign := 1; {将sign置1，如果是+的话正确，否则sign=-1，如果没有的话也是1}

s := 0;

if ch = '+'

then nextch

else if ch = '-'

then begin

nextch;

sign := -1

end;

if not(( ch >= '0' )and (ch <= '9' )) {如果不是数字报错}

then error( 40 )

else repeat {如果是数字重复}

s := 10\*s + ord( ord(ch)-ord('0')); {计算数字大小，直到遇到非数字}

nextch;

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

e := s\*sign + e {计算得出最终的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 {如果指数超上界，报错}

then error(21)

else if k + e < emin {如果超下界，说明精度不足，该实数太小，直接记为0}

then rnum := 0

else begin

s := abs(e);

t := 1.0;

d := 10.0;

repeat

while not odd(s) do {如果指数为偶数，直接提出一个平方，为了加快运算速度}

begin

s := s div 2;

d := sqr(d)

end;

s := s - 1; {奇数的时候，将上述得到的D乘给t}

t := d \* t {综合起来，就是转化为二进制，从低位开始往上算，比较快}

until s = 0;

if e >= 0

then rnum := rnum \* t {正的\*，负的/}

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 = '+'; {pascal中变量形参传地址，所以将b=+传回}

if not b

then if not( ch = '-' )

then begin { print error message } {如果不是+-的话，读到不是\*和，为止}

while( ch <> '\*' ) and ( ch <> ',' ) do

nextch;

end

else nextch

else nextch

end { switch };

begin { options }

repeat {重复，直到ch不是逗号为止}

nextch;

if ch <> '\*' {如果是\*的话跳过}

then begin

if ch = 't' {如果是t}

then begin

nextch;

switch( prtables ) {如果开关是t，处理并将返回值存入prtables}

end

else if ch = 's'

then begin

nextch;

switch( stackdump ) {如果是s，处理并将返回值存入stackdump}

end;

end

until ch <> ','

end { options };

begin { insymbol } {insymbol中的遗留问题，label}

1: while( ch = ' ' ) or ( ch = chr(9) ) do {chr是pascal自带函数，将数字转化为字符，所以就是跳过空格和tab}

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': {如果ch以字母开头，那就当单词识别}

begin { identifier of wordsymbol }

k := 0;

id := ' '; {将id设为10字节的空串}

repeat

if k < alng {如果没有超过限定的关键词长度，alng=10}

then begin

k := k + 1;

id[k] := ch

end;

nextch

until not((( ch >= 'a' ) and ( ch <= 'z' )) or (( ch >= '0') and (ch <= '9' ))); {直到不是字母或数字为止}

i := 1;

j := nkw; { binary search } {从保留字中做二分查找}

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] {如果找到了，那sy就是这一种key word}

else sy := ident {如果没找到，说明该单词是标识符}

end;

'0','1','2','3','4','5','6','7','8','9': {如果是以数字作为开始的，那么就是数字}

begin { number }

k := 0;

inum := 0;

sy := intcon; {sy暂定为整数}

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 ) {如果整数超出上限了，报错}

then begin

error(21);

inum := 0;

k := 0

end;

if ch = '.' {如果有一个点}

then begin

nextch;

if ch = '.' {两个点变成..运算}

then ch := ':'

else begin {如果只有一个点，说明确定是实数}

sy := realcon; {下面开始读取小数部分}

rnum := inum; {整数部分赋值}

e := 0;

while ( ch >= '0' ) and ( ch <= '9' ) do {将小数转化为科学计数法}

begin

e := e - 1;

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

nextch

end;

if e = 0 {如果小数点后面没有数字，那么报错}

then error(40);

if ch = 'e' {如果还有指数部分，那么读指数}

then readscale;

if e <> 0 then adjustscale {如果指数不为0，那么将科学计数法转化为实数}

end

end

else if ch = 'e' {如果没有小数部分，直接运算指数部分}

then begin

sy := realcon;

rnum := inum;

e := 0;

readscale;

if e <> 0

then adjustscale

end;

end;

':': {如果读到冒号}

begin

nextch;

if ch = '=' {如果是：=，那么就是赋值符号}

then begin

sy := becomes;

nextch

end

else sy := colon {如果只有冒号，那就是冒号}

end;

'<': {如果读到小于号}

begin

nextch;

if ch = '=' {如果是<=就是小于等于}

then begin

sy := leq;

nextch

end

else

if ch = '>' {如果是<>就是不等于}

then begin

sy := neq;

nextch

end

else sy := lss {否则就是小于}

end;

'>': {同理读到大于号}

begin

nextch;

if ch = '=' {如果是大于等于}

then begin

sy := geq;

nextch

end

else sy := gtr {否则就是大于}

end;

'.': {如果读到.}

begin

nextch;

if ch = '.' {如果两个..}

then begin

sy := colon; {那么就是冒号}

nextch

end

else sy := period {一个点就是句号}

end;

'''': {如果是单引号}

begin

k := 0;

2: nextch;

if ch = '''' {如果两个单引号}

then begin

nextch;

if ch <> '''' {如果不是3个，就跳到3}

then goto 3

end;

if sx + k = smax {字符常量需要存入表中，如果超了界就要报溢出错误}

then fatal(7);

stab[sx+k] := ch;

k := k + 1;

if cc = 1 {？cc=1为什么end of line}

then begin { end of line }

k := 0;

end

else goto 2;

3: if k = 1 {如果两个单引号中只有一个字符}

then begin

sy := charcon; {那就是字符类型}

inum := ord( stab[sx] ) {inum中存储该字符的ASCII码值}

end

else if k = 0 {如果是空引号}

then begin

error(38); {报错？但是为什么还存进去}

sy := charcon;

inum := 0

end

else begin {如果大于1那就是字符串了}

sy := stringcon; {字符串类型}

inum := sx; {位置}

sleng := k; {长度}

sx := sx + k {新的位置}

end

end;

'(': {如果是左小括号}

begin

nextch;

if ch <> '\*' {如果不是\*，那就是左小括号}

then sy := lparent

else begin { comment } {如果是(\*，就是注释}

nextch;

if ch = '$' {如果还有$，那就进行编译选项处理}

then options;

repeat

while ch <> '\*' do nextch;

nextch

until ch = ')'; {如果读到\*)，说明注释结束，返回1}

nextch;

goto 1

end

end;

'{': {如果是左大括号}

begin

nextch;

if ch = '$' {如果有$，进行编译选项处理}

then options;

while ch <> '}' do

nextch;

nextch;

goto 1 {注释结束返回1}

end;

'+', '-', '\*', '/', ')', '=', ',', '[', ']', ';': {如果是这些符号，有特殊的集合处理，以ch作为索引}

begin

sy := sps[ch];

nextch

end;

'$','"' ,'@', '?', '&', '^', '!': {如果单独出现这些符号，报错}

begin

error(24);

nextch;

goto 1

end

end { case }

end { insymbol };

procedure enter(x0:alfa; x1:objecttyp; x2:types; x3:integer ); {将分程序外的符号，包括标准类型、过程和函数的名字，存到符号表table中}

{需要传入前10个字符，类型，种类，值或地址}

begin

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

with tab[t] do {写入table[t]中}

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}

{需要元素种类，上下界}

begin

if l > h {如果下界大于上界报错}

then error(27);

if( abs(l) > xmax ) or ( abs(h) > xmax ) {如果上下界超界也报错}

then begin

error(27);

l := 0;

h := 0;

end;

if a = amax {如果数组表满了，报错}

then fatal(4)

else begin

a := a + 1; {如果都正确，数组计数+1}

with atab[a] do {写入atable}

begin

inxtyp := tp;

low := l;

high := h

end

end

end { enterarray };

procedure enterblock; {登录分程序表btable}

begin

if b = bmax {如果分程序表满了，报错}

then fatal(2)

else begin

b := b + 1; {否则，计数+1，写入btable}

btab[b].last := 0;

btab[b].lastpar := 0;

end

end { enterblock };

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

begin

if c2 = c2max - 1 {如果实常数表满了，报错}

then fatal(3)

else begin {实常数表有两个指针，c2用于存放位置，c1用来进行重复查询，以找空位，所以表中最多有一个数字重复，下一次覆盖写入}

rconst[c2+1] := x;

c1 := 1;

while rconst[c1] <> x do

c1 := c1 + 1;

if c1 > c2

then c2 := c1

end

end { enterreal };

procedure emit( fct: integer ); {生成P代码指令，这是第一种，只有func，没有xy}

begin

if lc = cmax {如果指令满了报错}

then fatal(6);

code[lc].f := fct;

lc := lc + 1

end { emit };

procedure emit1( fct, b: integer ); {第二种，有func和y，没有x}

begin

if lc = cmax {满了报错，不满写入}

then fatal(6);

with code[lc] do

begin

f := fct;

y := b;

end;

lc := lc + 1

end { emit1 };

procedure emit2( fct, a, b: integer ); {第三种，有func和x和y}

begin

if lc = cmax then fatal(6); {满了报错，不满写入}

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、实常数表和P-code}

var i: integer;

o: order;

mne: array[0..omax] of

packed array[1..5] of char; {mne最多存放omax条P-code，长度为5，因为三位指令+2位空格}

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); {空三行}

writeln(psout);

writeln(psout,' identifiers link obj typ ref nrm lev adr');

writeln(psout);

for i := btab[1].last to t do {从第一个分程序表的last，也就是最后一个标识符，到t个，输出符号表}

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' );

writeln( psout );

for i := 1 to b do {从1到到b输出分程序表}

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');

writeln( psout );

for i := 1 to a do {从1到a输出数组表}

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:');

writeln( psout );

for i := 0 to lc-1 do {从0到lc-1输出P-code}

begin

write( psout, i:5 );

o := code[i];

write( psout, mne[o.f]:8, o.f:5 ); {分类，因为有的Func没有x和y}

if o.f < 31

then if o.f < 4 {有x和y的就输出，没有的就不输出}

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 )

end { printtables };

procedure block( fsys: symset; isfun: boolean; level: integer ); {分析处理分程序，参数为symbol集合、是否为函数标识和分程序层数}

type conrec = record {根据tp类型建不同的record}

case tp: types of

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

reals :( r:real )

end;

var dx : integer ; { data allocation index } {声明四种变量}

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，直至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); {报错}

skipflag := true; {跳过flag=true}

while not ( sy in fsys ) do {如果sy不在fsys中，那就调用读取单词过程，直到在为止}

insymbol;

if skipflag then endskip {如果skipflag为真，那就执行上述endskip过程，打印错误}

end { skip };

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

begin

if not( sy in s1 ) {如果当前符号sy不再s1中，那么s1+s2合法集和停止集组成新的fsys，并调用skip，来跳过出错代码}

then skip( s1 + s2, n )

end { test };

procedure testsemicolon; {检查当前符号是否为分号}

begin

if sy = semicolon

then insymbol

else begin {如果不是分号报错}

error(14);

if sy in [comma, colon] {看看是不是逗号或者冒号，如果是的话，再读一个单词}

then insymbol

end;

test( [ident] + blockbegsys, fsys, 6 ) {检查新单词的合法性}

end { testsemicolon };

procedure enter( id: alfa; k:objecttyp ); {将分程序内的说明部分出现的符号装入符号表}

var j,l : integer;

begin

if t = tmax {如果符号表满了报错}

then fatal(1)

else begin

tab[0].name := id;

j := btab[display[level]].last; {将分程序表中level层最后一个标识符的位置给j}

l := j;

while tab[j].name <> id do {将标识符按链表找回，如果没有重名就继续找}

j := tab[j].link;

if j <> 0 {如果没有到0，说明有重复，报错}

then error(1)

else begin {否则将新的符号表项写入，与分程序外略有不同}

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

end

end

end { enter };

function loc( id: alfa ):integer;{查找标识符在符号表中的位置，返回值为integer，参数为10位名称}

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

begin

i := level; {i=分程序的层数}

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

repeat

j := btab[display[i]].last; {从display的第i层开始找，链表逐步找到头，如果第i层没有，到上一层找}

while tab[j].name <> id do

j := tab[j].link;

i := i - 1;

until ( i < 0 ) or ( j <> 0 ); {直到找遍或者找到为止}

if j = 0 {如果j=0，说明没找到，报错}

then error(0);

loc := j {否则返回位置j}

end { loc } ;

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

begin

if sy = ident {如果sy是标识符类型，那么使用enter，类型为变量}

then begin

enter( id, vvariable );

insymbol

end

else error(2) {否则报错}

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}

{conrec这个record在上面定义过}

var x, sign : integer;

begin

c.tp := notyp; {种类置为none type}

c.i := 0;{数值置为0}

test( constbegsys, fsys, 50 ); {看看当前符号是否合法，constbegsys中包括intcon，realcon和charcon + minus plus + ident}

if sy in constbegsys {如果sy在constbegsys中}

then begin

if sy = charcon {如果是char常量的话，写入c，tp是char类型，i为ASCII码}

then begin

c.tp := chars;

c.i := inum;

insymbol

end

else begin {如果不是char，那就是数字常量}

sign := 1;

if sy in [plus, minus] {如果是+/-正负号}

then begin

if sy = minus

then sign := -1;

insymbol {继续读一个分词出来}

end;

if sy = ident {如果是标识符的话？常量为什么有标识符}

then begin

x := loc(id); {那就找到标识符的位置}

if x <> 0

then

if tab[x].obj <> konstant {如果标识符在符号表中不是常数类型，报错}

then error(25)

else begin

c.tp := tab[x].typ; {如果是的话，tp = x的typ}

if c.tp = reals {如果是实数，将值存入r中不是i}

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

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

end;

insymbol

end

else if sy = intcon {如果是整数的话}

then begin

c.tp := ints; {tp为ints，i为符号\*inum数值}

c.i := sign\*inum;

insymbol

end

else if sy = realcon {如果是实数}

then begin

c.tp := reals; {tp为reals}

c.r := sign\*rnum; {r为符号\*rnum}

insymbol

end

else skip(fsys,50) {跳过没有用的符号}

end;

test(fsys,[],6)

end

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ref

和该类型大小sz}

var eltp : types; {变量定义}

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; {记录数组中元素的类型}

low, high : conrec; {记录数组的上下界}

elrf, elsz: integer; {临时记录ref和sz}

begin

constant( [colon, rbrack, rparent, ofsy] + fsys, low ); {constant获取常量，并由low返回获得数组编号的下界}

if low.tp = reals {如果下界是实数，则报错}

then begin

error(27);

low.tp := ints; {把low的类型转为ints，值存为0}

low.i := 0

end;

if sy = colon {如果symbol是冒号，也就是..，那么再识别下一个单词}

then insymbol

else error(13); {如果不是..的话报错}

constant( [rbrack, comma, rparent, ofsy ] + fsys, high ); {同理，获取数组的上界，由high存储返回值}

if high.tp <> low.tp {如果上界和下界的类型不等的话，也就是说，上界如果不是整数，报错}

then begin

error(27);

high.i := low.i

end;

enterarray( low.tp, low.i, high.i ); {将数组信息登录到atable中}

aref := a; {获取当前数组在atable中的位置}

if sy = comma {如果后面接逗号，说明需要建立多维数组}

then begin

insymbol; {那么就再读一个字符，把数组中的元素类型修改为arrays，并且递归调用自身处理数组元素}

eltp := arrays;

arraytyp( elrf, elsz )

end

else begin {没有，说明就是1维}

if sy = rbrack {遇到右中括号，则index部分声明完毕}

then insymbol

else begin {如果没有，则说明缺少右中括号，报错}

error(12);

if sy = rparent {如果是右小括号，容错继续}

then insymbol

end;

if sy = ofsy {继续获取of关键词}

then insymbol

else error(8); {如果没有of报错}

typ( fsys, eltp, elrf, elsz ) {递归调用自身，处理当前符号类型}

end;

with atab[aref] do {根据atable中的位置，记录当前数组的信息}

begin

arsz := (high-low+1) \* elsz; {计算该数组需要占用的存储空间}

size := arsz; {记录该数组需要占用的存储空间}

eltyp := eltp; {记录数组的元素类型}

elref := elrf; {记录数组在atab中登录的位置}

elsize := elsz {记录每个元素的大小}

end

end { arraytyp };

begin { typ } {开始类型处理过程}

tp := notyp; {用以存储变量的类型，初始为无类型}

rf := 0; {记录符号在符号表中的位置}

sz := 0; {记录该类型的大小}

test( typebegsys, fsys, 10 ); {测试当前符号是否是type声明的开始符号，如果不是报错}

if sy in typebegsys {如果sy在类型声明开始符号中}

then begin

if sy = ident {如果现在的符号是标识符}

then begin

x := loc(id); {根据id在符号表中查找重复定义}

if x <> 0 {！=0说明已经存在，对其符号表项进行操作}

then with tab[x] do

if obj <> typel {如果标识符的obj不是种类，报错}

then error(29)

else begin

tp := typ; {获得代表的类型}

rf := ref; {获得在符号表中的位置}

sz := adr; {获得在运行栈中分配的存储单元的相对地址}

if tp = notyp {如果是未定义类型，报错}

then error(30)

end;

insymbol

end

else if sy = arraysy {如果现在的符号是数组}

then begin

insymbol; {读一个找左中括号}

if sy = lbrack

then insymbol

else begin {如果不是左中括号报错，容错继续}

error(11);

if sy = lparent {找左小括号}

then insymbol

end;

tp := arrays; {说明tp是数组类型，调用arraytyp处理，获取atable中的登录位置和数组大小}

arraytyp(rf,sz)

end

else begin { records } {如果是record类型}

insymbol;

enterblock; {登录分程序表}

tp := records; {当前类型设为records}

rf := b; {rf指向当前过程在block表中的位置}

if level = lmax {如果嵌套层数达到最大，报表溢出错误}

then fatal(5);

level := level + 1; {否则level+1}

display[level] := b; {设置当前层次的display区，建立分层次索引}

offset := 0;

while not ( sy in fsys - [semicolon,comma,ident]+ [endsy] ) do {继续读取直到晕倒end}

begin { field section }

if sy = ident {如果sy是标识符}

then begin

t0 := t; {获取当前符号表table指针的位置}

entervariable; {将变量名登录到符号表中}

while sy = comma do {如果sy是逗号，说明同种变量中间用逗号分隔，则继续登录变量}

begin

insymbol;

entervariable

end;

if sy = colon {如果是分号，再读一个}

then insymbol

else error(5); {如果没有逗号也不是分号，说明格式有错误，报错}

t1 := t; {记录当前table中的指针位置，现在将t0与t1间的变量填入了，但是typ，ref等还没填写}

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，

确定各成员确定各成员的类型,ref和adr(对于不同的类型,ref和adr可以有不同存储含义)}

while t0 < t1 do {完成上面的填写工作}

begin

t0 := t0 + 1;

with tab[t0] do

begin

typ := eltp; {种类和指针}

ref := elrf;

normal := true;

adr := offset; {记录该变量相对于起始地址的位移}

offset := offset + elsz {更新offset给下一个成员变量使用}

end

end

end; { sy = ident }

if sy <> endsy {如果没有遇到end}

then begin

if sy = semicolon {如果sy是分号的话，继续}

then insymbol

else begin

error(14); {不是end也没有分号，报错}

if sy = comma {对逗号容错}

then insymbol

end;

test( [ident,endsy, semicolon],fsys,6 ) {测试当前符号的合法性，如果不合法，跳读}

end

end; { field section }

btab[rf].vsize := offset; {offset记录了所有变量的占用空间,将其记录下来}

sz := offset; {储存其占用空间总数}

btab[rf].psize := 0; {该程序块的参数占用空间设为0,因为record类型并不是真正的过程变量,没有参数}

insymbol; {后去下一个sym}

level := level - 1 {record声明结束后退出当前层次}

end; { record }

test( fsys, [],6 )

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; {记录形参的类型}

valpar : boolean; {记录当前形参是否为值形参}

rf, sz, x, t0 : integer;

begin

insymbol;

tp := notyp; {初始化为无类型}

rf := 0; {初始化符号表的位置}

sz := 0; {初始化元素大小}

test( [ident, varsy], fsys+[rparent], 7 ); {检验形参中的符号合法性}

while sy in [ident, varsy] do {如果sy是标识符或者变量}

begin

if sy <> varsy {如果不是变量标识}

then valpar := true {说明传值}

else begin

insymbol;

valpar := false {否则传地址}

end;

t0 := t; {记录当前符号表table栈顶位置}

entervariable; {将变量名登录到符号表中}

while sy = comma do {如果是逗号，说明一次好多同种变量}

begin

insymbol;

entervariable;

end;

if sy = colon {如果是冒号，该载入种类}

then begin

insymbol;

if sy <> ident {再读一个单词，如果不是标识符报错}

then error(2)

else begin

x := loc(id); {否则，找到该标识符的定义位置}

insymbol;

if x <> 0 {如果找到了，那么进行相应操作}

then with tab[x] do

if obj <> typel {如果当前的符号不是类型标识符，报错}

then error(29)

else begin

tp := typ; {获取参数的类型}

rf := ref; {获取参数在当前符号表的位置}

if valpar {如果是值形参}

then sz := adr {sz获得当前形参在符号表中的位置}

else sz := 1 {否则将sz置为1}

end;

end;

test( [semicolon, rparent], [comma,ident]+fsys, 14 ) {验证分号和右括号，如果有错报错并跳过}

end

else error(5); {没有冒号报错}

while t0 < t do {因为t0到t都是同一种类型，所以对应的一次填好}

begin

t0 := t0 + 1;

with tab[t0] do

begin

typ := tp;

ref := rf;

adr := dx;

lev := level;

normal := valpar;

dx := dx + sz

end

end;

if sy <> rparent {如果sy不是右小括号，继续判断}

then begin

if sy = semicolon {如果sy是分号，再读一个}

then insymbol

else begin {既不是右小括号也不是分号，报错，容错}

error(14);

if sy = comma {对逗号容错}

then insymbol

end;

test( [ident, varsy],[rparent]+fsys,6) {再判断是不是新的类型}

end

end { while }; {循环结束}

if sy = rparent {如果是右小括号}

then begin

insymbol;

test( [semicolon, colon],fsys,6 ) {判断是不是分号或者冒号}

end

else error(4) {不是右小括号的报错}

end { parameterlist }; {分析列表完毕}

procedure constdec; {处理常量定义，将常量名及其符号信息填入符号表}

var c : conrec; {存储类型和值}

begin

insymbol;

test([ident], blockbegsys, 2 ); {检查是不是标识符}

while sy = ident do {如果是标识符}

begin

enter(id, konstant); {登录到符号表当中，类型为常量}

insymbol;

if sy = eql {如果符号是等号}

then insymbol

else begin

error(16); {如果不是报错}

if sy = becomes {对赋值符号的容错}

then insymbol

end;

constant([semicolon,comma,ident]+fsys,c); {处理程序中出现的向量，并由c传回类型和数值}

tab[t].typ := c.tp; {将该常量写入符号表中}

tab[t].ref := 0;

if c.tp = reals {如果是实数}

then begin

enterreal(c.r); {则登录到实常数表rconst中}

tab[t].adr := c1;

end

else tab[t].adr := c.i; {否则是整数，直接table中赋值即可}

testsemicolon {测试当前符号是否是分号，以表示结束}

end

end { constdec };

procedure typedeclaration; {处理类型定义，并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 ); {检验是不是标识符}

while sy = ident do {如果是标识符的话}

begin

enter(id, typel); {类型名称和“类型”类型填入符号表}

t1 := t; {获取符号表栈顶指针}

insymbol;

if sy = eql {再读一个，如果是等号的话}

then insymbol

else begin

error(16);

if sy = becomes {对赋值符号容错}

then insymbol

end;

typ( [semicolon,comma,ident]+fsys, tp,rf,sz );{获得类型变量的类型,在符号表中的位置以及占用空间的大小}

with tab[t1] do {填入符号表中}

begin

typ := tp;

ref := rf;

adr := sz

end;

testsemicolon {判断是否分号结束}

end

end { typedeclaration };

procedure variabledeclaration; {处理变量声明}

var tp : types;

t0, t1, rf, sz : integer;

begin

insymbol;

while sy = ident do {读一个单词，如果是标识符的话}

begin

t0 := t; {记录当前位置}

entervariable; {将变量名登录到符号表中}

while sy = comma do {如果是逗号，继续登入同类型变量}

begin

insymbol;

entervariable;

end;

if sy = colon {如果是冒号，接类型定义}

then insymbol

else error(5); {没有的话报错}

t1 := t; {记录新的栈顶位置}

typ([semicolon,comma,ident]+fsys, tp,rf,sz ); {检验分号，逗号和标识符，获取类型、地址和大小}

while t0 < t1 do {一个一个登录}

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 {最后检验分号结束}

end

end { variabledeclaration };

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

var isfun : boolean;

begin

isfun := sy = funcsy; {isfun存储sy是不是函数类型}

insymbol;

if sy <> ident {如果不是标识符的话}

then begin

error(2); {报错，给id10个char}

id :=' '

end;

if isfun {如果是函数的话}

then enter(id,funktion) {在符号表中登录id和类型}

else enter(id,prozedure); {不是函数就是过程}

tab[t].normal := true;

insymbol;

block([semicolon]+fsys, isfun, level+1 ); {分析处理子程序，传入是否为程序，和深度}

if sy = semicolon {如果是分号}

then insymbol {继续}

else error(14); {否则报错}

emit(32+ord(isfun)) {exit} {生成过程或者函数的P-code，ord为char转ASCII}

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; {用于处理表达式的子程序,由x返回结果,forward使得selector可以调用expression}

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}

var x : item;

a,j : integer;

begin { sy in [lparent, lbrack, period] } {当前的符号应该是左括号,左分号或句号之一}

repeat

if sy = period {如果是点，因为引用成员变量的方式为'记录名.成员名',因此识别到'.'之后应该开始处理后面的结构名称}

then begin

insymbol; { field selector }

if sy <> ident {如果sy不是标识符，报错}

then error(2)

else begin

if v.typ <> records {如果处理的不是记录类型，报错}

then error(31)

else begin { search field identifier } {因为record把定义存放在btable中}

j := btab[v.ref].last; {j指向record中最后一个变量}

tab[0].name := id; {暂存当前符号的id}

while tab[j].name <> id do {在符号表中寻找当前符号}

j := tab[j].link; {没对应上则继续向前找}

if j = 0 {在record中没找到,说明符号未声明，报错}

then error(0);

v.typ := tab[j].typ; {找到了则获取属性}

v.ref := tab[j].ref; {记录其所在的btab位置}

a := tab[j].adr; {记录该成员变量相对于记录变量起始地址的位移}

if a <> 0 {如果位移不为零}

then emit1(9,a) {9对应INT初始化指令}

end;

insymbol

end

end

else begin { array selector } {如果不是点的话}

if sy <> lbrack {如果不是左中括号，报错}

then error(11);

repeat

insymbol;

expression( fsys+[comma,rbrack],x); {递归调用处理表达式的过程处理数组,获得返回结果保存到x中}

if v.typ <> arrays {如果type类型不是数组的话，报错}

then error(28)

else begin {否则，v是数组}

a := v.ref; {获取该数组在atable中的位置}

if atab[a].inxtyp <> x.typ {如果传入的下标和数组规定的下标类型不符，报错}

then error(26)

else if atab[a].elsize = 1 {如果元素的大小=1}

then emit1(20,a) {IDX，取元素下标地址}

else emit1(21,a); {IXX，取元素下标地址}

v.typ := atab[a].eltyp; {获得当前数组元素的类型}

v.ref := atab[a].elref {获得数组元素在atab中的位置}

end

until sy <> comma; {如果读到的不是逗号,说明只是1维数组}

if sy = rbrack {如果sy是右中括号}

then insymbol

else begin

error(12); {不是的话，报错}

if sy = rparent {对右小括号容错}

then insymbol

end

end

until not( sy in[lbrack, lparent, period]); {直到当前符号不是左括号,左分号或句号为止}

test( fsys,[],6) {检查当前符号是否合法}

end { selector };

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

var x : item;

lastp,cp,k : integer;

begin

emit1(18,i); { mark stack } {MKS，生成标记栈指令,传入被调用过程或函数在tab表中的位置}

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; {被调用过程或函数在符号表中的位置}

if sy = lparent {如果是左小括号}

then begin { actual parameter list } {开始参数列表分析}

repeat

insymbol;

if cp >= lastp {cp为被调用的过程的符号表中的位置，cp小于lastp，说明还有未声明的，正确执行；否则报错，参数太多了}

then error(39) {实参个数与形参个数不等}

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); {递归调用处理表达式的过程处理参数}

if x.typ = tab[cp].typ {如果参数的类型和符号表中规定的类型一致}

then begin

if x.ref <> tab[cp].ref {如果表达式指向的btable和符号表中所记录的btable不同，报错}

then error(36)

else if x.typ = arrays {如果参数是数组的话}

then emit1(22,atab[x.ref].size) {LDB，生成装入块指令，传参为数组的大小}

else if x.typ = records {如果参数是记录类型}

then emit1(22,btab[x.ref].vsize) {生成装入之灵}

end

else if ( x.typ = ints ) and ( tab[cp].typ = reals ) {如果x是整形，并且符号表中记录为实型}

then emit1(26,0) {FLT，转换浮点数}

else if x.typ <> notyp {如果参数无类型，报错}

then error(36);

end

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

if sy <> ident {如果sy不是标识符的话，报错}

then error(2)

else begin {否则，定位}

k := loc(id);

insymbol;

if k <> 0 {如果找到}

then begin

if tab[k].obj <> vvariable {如果同名标识符类型不是变量，报错}

then error(37);

x.typ := tab[k].typ; {记录当前符号的类型，并记录当前符号指向btable中的位置}

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

if tab[k].normal {如果是值形参}

then emit2(0,tab[k].lev,tab[k].adr) {LDA，把变量地址装入栈顶}

else emit2(1,tab[k].lev,tab[k].adr); {LOD，装入值}

if sy in [lbrack, lparent, period] {如果是左小括号-容错，左中括号-数组下标，句号-record成员}

then selector(fsys+[comma,colon,rparent],x); {调用分析子程序的过程}

if ( x.typ <> tab[cp].typ ) or ( x.ref <> tab[cp].ref ) {如果参数的符号类型不对，或者在表中的

位置和符号表中记录不同，均报错}

then error(36)

end

end

end {variable parameter }

end;

test( [comma, rparent],fsys,6) {检查逗号和右括号}

until sy <> comma; {如果不是逗号，说明参数声明结束了}

if sy = rparent {如果接右小括号，正确；否则报错}

then insymbol

else error(4)

end;

if cp < lastp {如果当前符号没有到最后一个符号的位置的话，报错，说明提供的参数太少了}

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

emit1(19,btab[tab[i].ref].psize-1 ); {CAL，调用用户过程}

if tab[i].lev < level {如果符号所在层次小于当前层次的话，需要更新display区，DIS}

then emit2(3,tab[i].lev, level )

end { call };

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

begin

if ( a > reals ) or ( b > reals ) {如果有操作数超过上限，报错}

then begin

error(33);

resulttype := notyp {返回nottype}

end

else if ( a = notyp ) or ( b = notyp ) {两个操作数中有一个nottype}

then resulttype := notyp {结果返回nottype}

else if a = ints {如果a是整形}

then if b = ints {如果b也是整型}

then resulttype := ints{结果就是整型}

else begin

resulttype := reals; {否则，结果是实型，需要FLT，浮点类型转换}

emit1(26,1)

end

else begin

resulttype := reals; {如果a是实型，结果就是实型}

if b = ints {如果b是整型，需要FLT，浮点类型转换}

then emit1(26,0)

end

end { resulttype } ;

procedure expression( fsys: symset; var x: item ); {处理表达式，返回类型和值}

var y : item;

op : symbol;

procedure simpleexpression( fsys: symset; var x: item ); {处理简单表达式}

var y : item;

op : symbol;

procedure term( fsys: symset; var x: item ); {处理项}

var y : item;

op : symbol;

procedure factor( fsys: symset; var x: item ); {处理因子}

var i,f : integer;

procedure standfct( n: integer ); {处理标准函数的子过程，传入编号为n}

var ts : typset;

begin { standard function no. n }

if sy = lparent {如果是左小括号}

then insymbol

else error(9);

if n < 17 {如果编号小于17}

then begin

expression( fsys+[rparent], x ); {递归调用处理表达式的过程处理参数，x获取参数信息}

case n of

{ abs, sqr } 0,2: begin {绝对值和平方}

ts := [ints, reals]; {ts为符号集合，定义为整型和实型}

tab[i].typ := x.typ; {参数的类型赋值}

if x.typ = reals {如果参数是实型，变成1和3}

then n := n + 1

end;

{ odd, chr } 4,5: ts := [ints]; {判断奇数和ASCII码转字符，允许符号为整型}

{ odr？ord吧？写错了 } 6: ts := [ints,bools,chars]; {字符转ASCII码，允许符号为整型、bool、和字符}

{ succ,pred } 7,8 : begin {后继函数和前继函数}

ts := [ints, bools,chars];

tab[i].typ := x.typ

end;

{ round,trunc } 9,10,11,12,13,14,15,16: {数学运算等}

{ sin,cos,... } begin

ts := [ints,reals]; {允许整型或实型}

if x.typ = ints {如果是整型，要FLT转化为浮点型}

then emit1(26,0)

end;

end; { case }

if x.typ in ts {如果参数的类型符合要求的符号集，FCT，执行标准程序}

then emit1(8,n)

else if x.typ <> notyp {如果不符合，且不是无符号类型}

then error(48);

end

else begin { n in [17,18] } {如果是判断输入是否结束的函数}

if sy <> ident {传入的不是标识符，报错}

then error(2)

else if id <> 'input ' {如果id不对，报错}

then error(0)

else insymbol;

emit1(8,n); {生成标准函数}

end;

x.typ := tab[i].typ; {记录参数的类型}

if sy = rparent {如果sy是右小括号，结束}

then insymbol

else error(4)

end { standfct } ;

begin { factor } {分析因子的子程序}

x.typ := notyp;

x.ref := 0;

test( facbegsys, fsys,58 ); {是不是因子的开始字符}

while sy in facbegsys do {如果是的话}

begin

if sy = ident {如果是标识符}

then begin

i := loc(id); {定位}

insymbol;

with tab[i] do

case obj of

konstant: begin {如果是常数}

x.typ := typ; {返回值的类型}

x.ref := 0; {索引值设置为0}

if x.typ = reals

then emit1(25,adr) {入栈}

else emit1(24,adr)

end;

vvariable:begin {如果是变量}

x.typ := typ; {获得返回值类型}

x.ref := ref; {获得返回值地址}

if sy in [lbrack, lparent,period] {如果是[(.，原因同上}

then begin

if normal {如果是传值}

then f := 0 {取地址}

else f := 1; {变量形参，取值放到栈顶}

emit2(f,lev,adr);

selector(fsys,x); {处理子结构}

if x.typ in stantyps {如果是标准类型}

then emit(34)

end

else begin {如果变量没有层次结构}

if x.typ in stantyps {如果是标准类型}

then if normal {如果传值}

then f := 1 {执行取值操作}

else f := 2 {否则间接取值}

else if normal {如果不是标准类型但是是值形参}

then f := 0 {取地址操作}

else f := 1; {都不是，取值}

emit2(f,lev,adr)

end

end;

typel,prozedure: error(44); {如果是类型类型或者过程类型,报错}

funktion: begin {如果是函数符号}

x.typ := typ;

if lev <> 0 {如果层次不为0,即不是标准函数}

then call(fsys,i) {调用call函数来处理函数调用}

else standfct(adr) {否则调用标准函数处理}

end

end { case,with }

end

else if sy in [ charcon,intcon,realcon ] {如果sy是常量类型}

then begin

if sy = realcon {如果是实数}

then begin

x.typ := reals; {将返回的type设置为实型}

enterreal(rnum); {将该实数登录实数表,rnum存有实数的值}

emit1(25,c1) {将实常量表中第c1个元素放入栈顶}

end

else begin

if sy = charcon {如果是字符}

then x.typ := chars

else x.typ := ints; {如果是整数}

emit1(24,inum)

end;

x.ref := 0;

insymbol

end

else if sy = lparent {如果是左括号}

then begin

insymbol;

expression(fsys + [rparent],x); {继续递归调用表达式处理，参数传回x}

if sy = rparent {如果右括号结束，否则报错}

then insymbol

else error(4)

end

else if sy = notsy {如果是非}

then begin

insymbol;

factor(fsys,x); {递归调用因子处理，参数回传x}

if x.typ = bools {如果非后是bool}

then emit(35) {NOT}

else if x.typ <> notyp {如果x无类型，报错？其他类型不报错？}

then error(32)

end;

test(fsys,facbegsys,6) {检查合法性}

end { while }

end { factor };

begin { term } {项处理过程}

factor( fsys + [times,rdiv,idiv,imod,andsy],x); {调用因子的分析程序开分析每一个因子项}

while sy in [times,rdiv,idiv,imod,andsy] do {如果因子后面跟符号\*/div mod and,说明后面还有因子,进入循环}

begin

op := sy; {记录操作符到op}

insymbol;

factor(fsys+[times,rdiv,idiv,imod,andsy],y );{递归调用因子分析程序，第二个参数存到y}

if op = times {如果是乘法}

then begin

x.typ := resulttype(x.typ, y.typ); {结果类型转换}

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 {如果实型除法}

then begin

if x.typ = ints {x如果整数}

then begin

emit1(26,1); {FLT}

x.typ := reals; {转实数}

end;

if y.typ = ints {y如果整数}

then begin

emit1(26,0); {FLT，转实数}

y.typ := reals;

end;

if (x.typ = reals) and (y.typ = reals) {如果都是实数，DIR，实型除}

then emit(61)

else begin

if( x.typ <> notyp ) and (y.typ <> notyp) {如果不是实数，说明类型转换失败，有错，报错}

then error(33);

x.typ := notyp {结果无效，无类型}

end

end

else if op = andsy {op为与}

then begin

if( x.typ = bools )and(y.typ = bools) {xy都是bool类型}

then emit(56) {AND逻辑与}

else begin

if( x.typ <> notyp ) and (y.typ <> notyp) {如果不是无类型}

then error(32); {报错}

x.typ := notyp {x结果无效，无类型}

end

end

else begin { op in [idiv,imod] } {如果是整型除法或取余}

if (x.typ = ints) and (y.typ = ints) {如果xy都是整数}

then if op = idiv

then emit(58) {整形除法DIV}

else emit(59) {取模MOD}

else begin

if ( x.typ <> notyp ) and (y.typ <> notyp) {都不是无类型，报错}

then error(34);

x.typ := notyp {结果无效}

end

end

end { while }

end { term }; {项处理结束}

begin { simpleexpression } {处理简单表达式}

if sy in [plus,minus] {如果是+号或者-号}

then begin

op := sy; {记录运算符}

insymbol;

term( fsys+[plus,minus],x); {调用项处理过程，+-不报错，返回值放x}

if x.typ > reals {如果类型是bools chars arrays records，报错}

then error(33)

else if op = minus {如果是减号}

then emit(36) {MUS，取相反数}

end

else term(fsys+[plus,minus,orsy],x); {+-or不报错，调用项处理过程}

while sy in [plus,minus,orsy] do {如果是+-or}

begin

op := sy; {取操作符}

insymbol;

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

if op = orsy {如果是or}

then begin

if ( x.typ = bools )and(y.typ = bools) {如果是两个bool类型}

then emit(51)

else begin

if( x.typ <> notyp) and (y.typ <> notyp) {如果都不是无类型}

then error(32);

x.typ := notyp {结果无效}

end

end

else begin

x.typ := resulttype(x.typ,y.typ); {不是or，说明是+-}

case x.typ of

notyp: ;

ints: if op = plus {如果是整数+ -}

then emit(52)

else emit(53);

reals:if op = plus {如果是实数+ -}

then emit(54)

else emit(55)

end { case }

end

end { while }

end { simpleexpression }; {简单表达式分析结束}

begin { expression } {表达式分析子程序}

simpleexpression(fsys+[eql,neq,lss,leq,gtr,geq],x); {简单子程序分析，同时可以识别= ！= < > <> <= >=}

if sy in [ eql,neq,lss,leq,gtr,geq] {如果sy是比较符号}

then begin

op := sy; {记录op}

insymbol;

simpleexpression(fsys,y); {调用简单分析子程序，第二个参数返回值为y}

if(x.typ in [notyp,ints,bools,chars]) and (x.typ = y.typ) {如果x的类型为无类型、整型、bool、字符，并且xy类型相同}

then case op of {根据op决定P-code}

eql: emit(45);

neq: emit(46);

lss: emit(47);

leq: emit(48);

gtr: emit(49);

geq: emit(50);

end

else begin {如果不等或者类型不属于那四种}

if x.typ = ints {如果x是整型}

then begin

x.typ := reals; {把x换成实型}

emit1(26,1)

end

else if y.typ = ints {如果y是整型}

then begin

y.typ := reals; {把y换成实型}

emit1(26,0)

end;

if ( x.typ = reals)and(y.typ=reals) {如果两个都是实型，说明还是对的}

then case op of {继续判断一遍}

eql: emit(39);

neq: emit(40);

lss: emit(41);

leq: emit(42);

gtr: emit(43);

geq: emit(44);

end

else error(35) {再不一样，就没办法了，就是真的不能比较了}

end;

x.typ := bools {x的结果变为bool型}

end

end { expression }; {表达式分析结束}

procedure assignment( lv, ad: integer ); {处理赋值语句的过程，lv是level，ad是地址}

var x,y: item;

f : integer;

begin { tab[i].obj in [variable,prozedure] } {当当前符号表的目标变量为变量或者过程型时}

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

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

if tab[i].normal {如果是传值}

then f := 0 {取地址}

else f := 1; {变量形参，取值放到栈顶}

emit2(f,lv,ad);

if sy in [lbrack,lparent,period] {如果是[(.，}

then selector([becomes,eql]+fsys,x); {需要处理下标}

if sy = becomes {如果是赋值}

then insymbol

else begin {否则报错}

error(51);

if sy = eql {对=容错}

then insymbol

end;

expression(fsys,y); {调用表达式分析程序，y=赋值符号右边的值}

if x.typ = y.typ

then if x.typ in stantyps {如果是标准类型}

then emit(38) {STO，将栈顶内容存入，以栈顶次高元素为地址的单元，赋值符号}

else if x.ref <> y.ref {如果表中指针不同，报错}

then error(46)

else if x.typ = arrays {如果是数组}

then emit1(23,atab[x.ref].size) {CPB，复制块，从atable中}

else emit1(23,btab[x.ref].vsize) {如果不是数组，那就是record了，从btable中}

else if(x.typ = reals )and (y.typ = ints) {如果类型不一致，类型转换}

then begin

emit1(26,0);

emit(38)

end

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

then error(46)

end { assignment };

procedure compoundstatement; {处理符合语句}

begin

insymbol;

statement([semicolon,endsy]+fsys); {递归调用语句处理}

while sy in [semicolon]+statbegsys do {如果是分号的话，或者是语句的开始字符}

begin

if sy = semicolon

then insymbol

else error(14);

statement([semicolon,endsy]+fsys)

end;

if sy = endsy {都读完了，该END了}

then insymbol

else error(57)

end { compoundstatement };

procedure ifstatement; {处理if语句}

var x : item;

lc1,lc2: integer;

begin

insymbol;

expression( fsys+[thensy,dosy],x); {表达式分析子程序}

if not ( x.typ in [bools,notyp]) {如果x的种类不是bool或者无类型，报错}

then error(17);

lc1 := lc; {记录当前程序位置}

emit(11); { jmpc，有条件跳转}

if sy = thensy {如果sy是then}

then insymbol

else begin

error(52); {不是的话，报错}

if sy = dosy {对缺少then，直接do容错}

then insymbol

end;

statement( fsys+[elsesy]); {读取else}

if sy = elsesy {如果有else}

then begin

insymbol;

lc2 := lc; {记录else的位置}

emit(10); {JMP，无条件跳转}

code[lc1].y := lc; {if的跳转位置放到else后面}

statement(fsys);

code[lc2].y := lc {把else的跳转位置放到statement后面}

end

else code[lc1].y := lc {如果没有else，把if的跳转放到statement前面}

end { ifstatement };

procedure casestatement; {处理case语句}

var x : item; {变量定义}

i,j,k,lc1 : integer;

casetab : array[1..csmax]of {casetab中放csmax个record，包括值和跳转地址}

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;

k : integer;

begin

constant( fsys+[comma,colon],lab ); {常量检测识别，返回值放到lab中}

if lab.tp <> x.typ {如果lab的类型和参数的类型不同，报错}

then error(47)

else if i = csmax {如果声明达到了case的最大限制，报错，表溢出}

then fatal(6)

else begin {正确的话}

i := i+1; {case的数量i}

k := 0;

casetab[i].val := lab.i; {赋个数和跳转位置}

casetab[i].lc := lc;

repeat

k := k+1 {查询索引}

until casetab[k].val = lab.i; {如果相等停下}

if k < i {如果重复，报错}

then error(1); { multiple definition }

end

end { caselabel };

procedure onecase; {处理case语句中的一个分支}

begin

if sy in constbegsys {如果sy是常量的开始字符}

then begin

caselabel; {调用caselabel，填表}

while sy = comma do {如果还有逗号，继续填表，说明是一个case对应多个标签}

begin

insymbol;

caselabel

end;

if sy = colon {如果不是冒号，报错}

then insymbol

else error(5);

statement([semicolon,endsy]+fsys); {检查分号和end}

j := j+1; {用来记录当前case对应exittab的位置}

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

emit(10){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 ); {递归调用处理表达式的方式先获得当前表达式的属性,即case后面变量的类型}

if not( x.typ in [ints,bools,chars,notyp ]) {如果当前的表达式不是整数,布尔型,字符型或未定义类型}

then error(23);

lc1 := lc; {记录当前PCODE代码的位置指针}

emit(12); {jmpx}

if sy = ofsy {如果sy为of}

then insymbol

else error(8);

onecase; {调用onecase处理子程序}

while sy = semicolon do {如果sy是分号，说明还有其他case分支}

begin

insymbol;

onecase

end;

code[lc1].y := lc; {回填之前条件跳转的位置，如果不对，跳到case最后}

for k := 1 to i do

begin

emit1( 13,casetab[k].val); {CAS，情况表登记项，伪指令，不能执行}

emit1( 13,casetab[k].lc);

end;

emit1(10,0); {生成跳转语句，说明case表介乎}

for k := 1 to j do

code[exittab[k]].y := lc; {把刚刚所有的case语句的跳转地址都修改为case最后}

if sy = endsy {如果不是end，报错}

then insymbol

else error(57)

end { casestatement };

procedure repeatstatement; {处理repeat语句的处理过程}

var x : item; {用来获取返回值}

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 {如果是分号，继续；否则报错}

then insymbol

else error(14);

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

end;

if sy = untilsy {如果sy是until}

then begin

insymbol;

expression(fsys,x); {处理该表达式,获得其类型}

if not(x.typ in [bools,notyp] ) {如果until后面不是bool类型或者无类型，报错}

then error(17);

emit1(11,lc1); {有条件跳转}

end

else error(53)

end { repeatstatement };

procedure whilestatement; {处理while语句}

var x : item;

lc1,lc2 : integer;

begin

insymbol;

lc1 := lc; {记录当前P-code位置}

expression( fsys+[dosy],x); {调用expression子程序处理while直到do，返回值给x}

if not( x.typ in [bools, notyp]) {如果x的类型不是bool或者无类型，报错}

then error(17);

lc2 := lc; {表达式之后的位置lc2}

emit(11); {有条件跳转}

if sy = dosy {如果是do，继续}

then insymbol

else error(54);

statement(fsys); {使用statement处理}

emit1(10,lc1); {无条件跳转回到while头}

code[lc2].y := lc {把while后的跳转放到statement之后}

end { whilestatement };

procedure forstatement; {处理for语句}

var cvt : types;

x : item;

i,f,lc1,lc2 : integer;

begin

insymbol;

if sy = ident {如果获取到的是标识符}

then begin

i := loc(id); {找到这个标识符在符号表中登陆的位置，}

insymbol;

if i = 0 {如果没有找到这个标识符}

then cvt := ints {计数变量类型默认为整形}

else if tab[i].obj = vvariable {如果对应的这个标识符对应符号的大类是变量类型}

then begin

cvt := tab[i].typ; {计数变量类型就设置为这个变量的类型}

if not tab[i].normal {如果是变量形参,即变量存储的是值而非地址，报错}

then error(37)

else emit2(0,tab[i].lev, tab[i].adr ); {如果不是变量类型, 获取该符号的地址，}

if not ( cvt in [notyp, ints, bools, chars]) {如果获取到计数变量的类型不是未定义,整型,布尔型,字符型，报错}

then error(18)

end

else begin {如果符号的类型也不是变量，直接报错}

error(37);

cvt := ints {将计数变量类型设置为整型}

end

end

else skip([becomes,tosy,downtosy,dosy]+fsys,2); {如果不是标识符，跳过并报错直到赋值 to downto do等}

if sy = becomes {如果sy是赋值}

then begin

insymbol;

expression( [tosy, downtosy,dosy]+fsys,x); {递归调用处理表达式的方式来获得表达式的值和类型}

if x.typ <> cvt {如果获取到的表达式类型和计数变量的符号类型不相同，报错}

then error(19);

end

else skip([tosy, downtosy,dosy]+fsys,51); {如果不是赋值，跳过并报错直到to downto do等}

f := 14; {指令编号暂存14}

if sy in [tosy,downtosy] {如果是to或者downto}

then begin

if sy = downtosy {如果是downto}

then f := 16; {指令编号暂存16}

insymbol;

expression([dosy]+fsys,x); {处理表达式}

if x.typ <> cvt {如果表达式的类型不是计数类型的话，报错}

then error(19)

end

else skip([dosy]+fsys,55); {skip直到do}

lc1 := lc; {记录当前位置}

emit(f); {根据上面的情况产生不同指令}

if sy = dosy {如果sy是do继续，不是报错}

then insymbol

else error(54);

lc2 := lc; {记录当前指令到lc2}

statement(fsys);

emit1(f+1,lc2); {增长或减小类for循环的再入入口}

code[lc1].y := lc {修改跳出位置为statement后}

end { forstatement };

procedure standproc( n: integer ); {处理标准输入或输出过程的调用}

var i,f : integer;

x,y : item;

begin

case n of

1,2 : begin { read }

if not iflag {如果没有iflag，遇到input，iflag=true}

then begin {报错，iflag= true}

error(20);

iflag := true

end;

if sy = lparent {如果是左小括号}

then begin

repeat

insymbol;

if sy <> ident {如果不是标识符，报错}

then error(2)

else begin

i := loc(id); {定位}

insymbol;

if i <> 0 {找到了}

then if tab[i].obj <> vvariable {如果种类不是变量的话，报错}

then error(37)

else begin

x.typ := tab[i].typ; {否则记录下类型和位置}

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

if tab[i].normal {如果值传递}

then f := 0

else f := 1; {参数传递}

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

if sy in [lbrack,lparent,period] {左中小括号和点}

then selector( fsys+[comma,rparent],x); {处理结构变量存到x}

if x.typ in [ints,reals,chars,notyp] {如果是这些类型}

then emit1(27,ord(x.typ)) {RED读}

else error(41) {否则报错}

end

end;

test([comma,rparent],fsys,6); {检查逗号和右小括号，处理无用字符}

until sy <> comma; {直到sy不是逗号}

if sy = rparent {如果sy是右小括号，继续，否则报错}

then insymbol

else error(4)

end;

if n = 2 {如果是2，读一行}

then emit(62)

end;

3,4 : begin { write }

if sy = lparent {如果是左小括号}

then begin

repeat {重复}

insymbol;

if sy = stringcon {如果是字符常量}

then begin

emit1(24,sleng); {LDC，装入字面常量}

emit1(28,inum); {WRS，写字符}

insymbol

end

else begin

expression(fsys+[comma,colon,rparent],x); {表达式处理，返回值给x}

if not( x.typ in stantyps ) {如果x不是标准类型，报错}

then error(41);

if sy = colon {如果下一个字符是冒号}

then begin

insymbol;

expression( fsys+[comma,colon,rparent],y); {表达式处理，返回值给y}

if y.typ <> ints {如果y不是整型，报错}

then error(43);

if sy = colon {如果sy是冒号}

then begin

if x.typ <> reals {如果x不是实型，报错}

then error(42);

insymbol;

expression(fsys+[comma,rparent],y); {表达式处理}

if y.typ <> ints {如果y不是整型，报错}

then error(43);

emit(37) {WRR，写实数}

end

else emit1(30,ord(x.typ)) {如果不是冒号，WRU，写}

end

else emit1(29,ord(x.typ))

end

until sy <> comma; {直到sy不是逗号}

if sy = rparent {如果sy是右括号}

then insymbol

else error(4) {不是，报错}

end;

if n = 4

then emit(63) {WRL，换行写}

end; { write }

end { case };

end { standproc } ; {标准输入输出结束}

begin { statement } {总的statement描述}

if sy in statbegsys+[ident] {如果是statement首字符或者标识符}

then case sy of

ident : begin {如果sy是标识符}

i := loc(id); {定位}

insymbol;

if i <> 0 {如果找到}

then case tab[i].obj of

konstant,typel : error(45); {如果是常量或者类型量，报错}

vvariable: assignment( tab[i].lev,tab[i].adr); {如果是变量，赋值}

prozedure: if tab[i].lev <> 0 {如果是过程，层数不=0，不是系统过程}

then call(fsys,i) {则call}

else standproc(tab[i].adr); {是系统过程，则standproc}

funktion: if tab[i].ref = display[level] {函数的话，层数如果对}

then assignment(tab[i].lev+1,0) {处理赋值语句}

else error(45) {否则报错}

end { case }

end;

beginsy : compoundstatement; {如果是begin的话，执行复杂语句}

ifsy : ifstatement; {如果是if，执行if语句}

casesy : casestatement; {如果是case，执行case语句}

whilesy : whilestatement; {执行while语句}

repeatsy: repeatstatement; {执行repeat语句}

forsy : forstatement; {执行for语句}

end; { case }

test( fsys, [],14); {跳过无用字符}

end { statement };

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; {获取当前符号表的位置}

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，报错，表溢出}

then fatal(5);

test([lparent,colon,semicolon],fsys,14); {检查当前的符号是否是左括号,冒号,分号中的一个,不是报错}

enterblock; {登录分程序表}

prb := b; {prb记录分程序表中的位置}

display[level] := b; {display的层数记为b}

tab[prt].typ := notyp; {修改符号表}

tab[prt].ref := prb;

if ( sy = lparent ) and ( level > 1 ) {如果sy是左括号，并且大于1层}

then parameterlist; {分析参数}

btab[prb].lastpar := t; {新block的最后一个参数指向t}

btab[prb].psize := dx;

if isfun {如果是函数的话}

then if sy = colon {找冒号看返回类型}

then begin

insymbol; { function type }

if sy = ident {如果是标识符的话}

then begin

x := loc(id); {定位}

insymbol;

if x <> 0 {如果找到了}

then if tab[x].typ in stantyps {如果是标准类型，类型赋值，如果不是，报错}

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

else error(15)

end

else skip( [semicolon]+fsys,2 ) {跳过分号}

end

else error(5); {如果没有冒号，报错}

if sy = semicolon {如果sy是分号，继续}

then insymbol

else error(14);

repeat{重复}

if sy = constsy {如果sy是常数，进行常数声明}

then constdec;

if sy = typesy {如果sy是类型，进行类型声明}

then typedeclaration;

if sy = varsy {如果sy是变量，进行变量声明}

then variabledeclaration;

btab[prb].vsize := dx;

while sy in [procsy,funcsy] do {如果是过程或者变量，进行过程声明}

procdeclaration;

test([beginsy],blockbegsys+statbegsys,56) {测试begin}

until sy in statbegsys; {如果sy在state首字符里}

tab[prt].adr := lc; {记录函数的开始地址}

insymbol;

statement([semicolon,endsy]+fsys); {调用statement}

while sy in [semicolon]+statbegsys do {如果sy为分号或首字符云鬟}

begin

if sy = semicolon {如果是分号继续，不是报错}

then insymbol

else error(14);

statement([semicolon,endsy]+fsys); {继续调用statement分析子程序}

end;

if sy = endsy {如果statement都处理完了，应该是end了}

then insymbol

else error(57);

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

end { block };

procedure interpret; {P-code解释执行程序}

var ir : order ; { instruction buffer } {当前指令的位置}

pc : integer; { program counter } {类似于指令寄存器}

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

b : integer; { base index } {基地址}

h1,h2,h3: integer; {三个临时变量}

lncnt,ocnt,blkcnt,chrcnt: integer; { counters } {四种计数器}

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

fld: array [1..4] of integer; { default field widths } {？默认域宽什么意思}

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

s : array[1..stacksize] of { blockmark: }

record

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;

procedure dump; {程序运行时，卸出打印现场剖析信息，display，t，b以及运行栈S的内容}

var p,h3 : integer;

begin

h3 := tab[h2].lev; {存储h2的level}

writeln(psout); {打印多个空行，以及文字提示}

writeln(psout);

writeln(psout,' calling ', tab[h2].name );

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);

writeln(psout);

writeln(psout);

writeln(psout,' top of stack ',t:4,' frame base ':14,b:4);

writeln(psout);

writeln(psout);

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

writeln(psout);

for p := t downto 1 do

writeln( psout, p:14, s[p].i:8);

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

end; {dump }

procedure inter0; {下面为不同P-Code所对应的操作}

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,以记录错误类型}

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 {检查栈是否溢出,溢出则报错}

then ps := stkchk

else s[t] := s[display[ir.x]+ir.y] {由于传入的是地址,完成取值后将值放到栈顶}

end;

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

t := t + 1;

if t > stacksize {检查栈是否溢出,溢出则报错}

then ps := stkchk

else s[t] := s[s[display[ir.x]+ir.y].i] {多一步，使用栈内地址的值作为引用}

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,ir.y是函数的编号,FCT}

0 : s[t].i := abs(s[t].i); {整数x求绝对值}

1 : s[t].r := abs(s[t].r); {实数x求绝对值}

2 : s[t].i := sqr(s[t].i); {整数x求平方}

3 : s[t].r := sqr(s[t].r); {实数x求平方}

4 : s[t].b := odd(s[t].i); {整数x判奇偶性,计数返回1}

5 : s[t].c := chr(s[t].i); {ascii码x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}

12 : s[t].r := cos(s[t].r); {求余弦sin(x),注意x为实数弧度}

13 : s[t].r := exp(s[t].r); {求e^x,x为实数}

14 : s[t].r := ln(s[t].r); {求自然对数ln(x),x为实数}

15 : s[t].r := sqrt(s[t].r); {实数x开方}

16 : s[t].r := arcTan(s[t].r); {反三角函数arctan(x)}

17 : begin

t := t+1;

if t > stacksize {溢出错误}

then ps := stkchk

else s[t].b := eof(prd) {判断输入有没有读完}

end;

18 : begin

t := t+1;

if t > stacksize {一处错误}

then ps := stkchk

else s[t].b := eoln(prd) {判断该行有没有读完}

end;

end;

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

end { case ir.y }

end; { inter0 }

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}

11 : begin { conditional jump } {条件跳转语句,JPC}

if not s[t].b {如果栈顶值为假}

then pc := ir.y; {跳转到y指令}

t := t - 1 {退栈}

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; {记录需要跳转到的地址}

h3 := 0;

repeat

if code[h2].f <> 13 {如果操作码不是13,证明跳转到的不是情况表}

then begin

h3 := 1;

ps := caschk

end

else if code[h2].y = h1

then begin

h3 := 1;

pc := code[h2+1].y

end

else h2 := h2 + 2

until h3 <> 0

end;

14 : begin { for1up} {增量步长for循环的初始判断,F1U}

h1 := s[t-1].i; {for循环之前需要储存计数变量的地址,初值和终值,这里h1获取的是初值}

if h1 <= s[t].i {如果初值小于等于终值}

then s[s[t-2].i].i := h1 {开始循环,将技术变量的值赋为初值}

else begin {否则循环完毕}

t := t - 3; {退栈3格,退去计数变量的地址,初值和终值所占用的空间}

pc := ir.y {跳出循环,注意这里的y是由后方语句回传得到的}

end

end;

15 : begin { for2up } {增量步长的结束判断,F2U}

h2 := s[t-2].i; {获得计数变量的地址}

h1 := s[h2].i+1; {h1为计数变量的值+1}

if h1 <= s[t].i {判断是否还满足循环条件}

then begin

s[h2].i := h1; {如果满足,将h1赋给计数变量}

pc := ir.y {跳转到循环的开始位置}

end

else t := t-3; {不满足的情况不做跳转(执行下一条),退栈3格}

end;

16 : begin { for1down } {减量步长for循环的初始判断,F1U}

h1 := s[t-1].i; {for循环之前需要储存计数变量的地址,初值和终值,这里h1获取的是初值}

if h1 >= s[t].i {如果初值大于等于终值}

then s[s[t-2].i].i := h1 {开始循环,将技术变量的值赋为初值}

else begin {否则循环完毕}

pc := ir.y; {跳出循环,注意这里的y是由后方语句回传得到的}

t := t - 3 {退栈3格,退去计数变量的地址,初值和终值所占用的空间}

end

end;

17 : begin { for2down } {减量步长的结束判断,F2U}

h2 := s[t-2].i; {获得计数变量的地址}

h1 := s[h2].i-1; {h1为计数变量的值-1}

if h1 >= s[t].i {判断是否还满足循环条件}

then begin

s[h2].i := h1; {如果满足,将h1赋给计数变量}

pc := ir.y {跳转到循环的开始位置}

end

else t := t-3; {不满足的情况不做跳转(执行下一条),退栈3格}

end;

18 : begin { mark stack } {标记栈}

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 {如果超过上限报错}

then ps := stkchk

else begin

t := t+5; {预留内务信息区}

s[t-1].i := h1-1; {次栈顶存放vsize-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}

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;

s[h1+1].i := pc; {栈基地址+1存pc}

s[h1+2].i := display[h3]; {+2存display区}

s[h1+3].i := b; {+3存btable指针}

for h3 := t+1 to h4 do

s[h3].i := 0;

b := h1;

t := h4;

pc := tab[h2].adr;

if stackdump {如果stackdump开关开着，执行dump}

then dump

end;

end { case }

end; { inter1 }

procedure inter2;

begin

case ir.f of

20 : begin { index1 } {IDX，取下标变量地址，元素长度=1}

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

h2 := atab[h1].low; {h2是h1的下界}

h3 := s[t].i; {h3是栈顶指针的值}

if h3 < h2 {如果h3小于下界，报错}

then ps := inxchk

else if h3 > atab[h1].high {如果h3大于上界，也报错}

then ps := inxchk

else begin {正确的话，栈顶指针-1，栈顶的值+差值\*1}

t := t-1;

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

end

end;

21 : begin { index } {IXX，取下标变量地址}

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

h2 := atab[h1].low; {h2是h1的下界}

h3 := s[t].i; {h3是栈顶指针的值}

if h3 < h2 {如果h3小于下界，报错}

then ps := inxchk

else if h3 > atab[h1].high {如果h3大于上界，也报错}

then ps := inxchk

else begin {正确的话，栈顶指针-1，栈顶的值+差值\*元素大小}

t := t-1;

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

end

end;

22 : begin { load block } {LDB，装入块}

h1 := s[t].i; {获取栈顶值}

t := t-1;

h2 := ir.y+t; {获取需要分配到的空间位置}

if h2 > stacksize {栈空间不足,报错}

then ps := stkchk

else while t < h2 do {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 } {CPB，复制块，将栈中一块向下平移1行}

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，LDC}

t := t+1;

if t > stacksize

then ps := stkchk

else s[t].i := ir.y {对于整型变量y直接装入栈顶}

end;

25 : begin { load real } {读取实数,LDR}

t := t+1;

if t > stacksize

then ps := stkchk

else s[t].r := rconst[ir.y] {将实常量表中第i个元素放到数据栈的栈顶}

end;

26 : begin { float } {整型转实型,FLT}

h1 := t-ir.y; {获得符号的地址}

s[h1].r := s[h1].i {令实型等于整数部分}

end;

27 : begin { read }

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 } {WRS写字符}

h1 := s[t].i; {h1栈顶数值}

h2 := ir.y; {h2=y}

t := t-1;

chrcnt := chrcnt+h1; {字符数量+1}

if chrcnt > lineleng {如果数量>最大输出大小}

then ps := lngchk; {报错}

repeat

write(prr,stab[h2]); {重复写h1个字符}

h1 := h1-1;

h2 := h2+1

until h1 = 0

end;

29 : begin { write1 } {WRW，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 } {WRU，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; {finish停止}

32 : begin { exit procedure } {退出过程,EXP}

t := b-1; {退栈}

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,EXF}

t := b; {退栈,要保留函数名}

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;

34 : s[t] := s[s[t].i]; {LDT，取站定单元内容为地址的单元内容}

35 : s[t].b := not s[t].b; {Not，逻辑非}

36 : s[t].i := -s[t].i; {MUS，求负数}

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 } {STO，将栈顶内容存入以次栈顶为地址的单元}

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

t := t-2 {退两个}

end;

39 : begin {EQR，实数比较相等}

t := t-1;

s[t].b := s[t].r=s[t+1].r {t-1栈顶，布尔值存放两个实数是否相等}

end;

end { case }

end; { inter3 }

procedure inter4;

begin

case ir.f of

40 : begin {NER，实数不等比较}

t := t-1;

s[t].b := s[t].r <> s[t+1].r {t-1栈顶，布尔值存放两个实数不相等}

end;

41 : begin {LSR，实数小于比较}

t := t-1;

s[t].b := s[t].r < s[t+1].r {t-1栈顶，布尔值存放两个实数是否小于}

end;

42 : begin {LER，实数小于等于}

t := t-1;

s[t].b := s[t].r <= s[t+1].r {t-1栈顶，布尔值存放两个实数是否小于等于}

end;

43 : begin {GTR，实数大于}

t := t-1;

s[t].b := s[t].r > s[t+1].r {t-1栈顶，布尔值存放两个实数是否大于}

end;

44 : begin {GER，实数大于等于}

t := t-1;

s[t].b := s[t].r >= s[t+1].r {t-1栈顶，布尔值存放两个实数是否大于等于}

end;

45 : begin {EQL，整数等于}

t := t-1;

s[t].b := s[t].i = s[t+1].i {t-1栈顶，布尔值存放两个整数是否相等}

end;

46 : begin {NEQ，整数不等}

t := t-1;

s[t].b := s[t].i <> s[t+1].i {t-1栈顶，布尔值存放两个整数是否不等}

end;

47 : begin {LSS，整数小于}

t := t-1;

s[t].b := s[t].i < s[t+1].i {t-1栈顶，布尔值存放两个整数是否小于}

end;

48 : begin {LER，整数小于等于}

t := t-1;

s[t].b := s[t].i <= s[t+1].i {t-1栈顶，布尔值存放两个整数是否小于等于}

end;

49 : begin {Grt，整数大于}

t := t-1;

s[t].b := s[t].i > s[t+1].i {t-1栈顶，布尔值存放两个整数是否大于}

end;

end { case }

end; { inter4 }

procedure inter5;

begin

case ir.f of

50 : begin {整型大于等于,GEQ}

t := t-1;

s[t].b := s[t].i >= s[t+1].i {t-1栈顶，布尔值存放两个整数是否大于等于}

end;

51 : begin {OR指令,ORR}

t := t-1;

s[t].b := s[t].b or s[t+1].b {t-1栈顶，布尔值存放两个bool或}

end;

52 : begin {整数加,ADD}

t := t-1;

s[t].i := s[t].i+s[t+1].i {t-1栈顶，int存放两个整数+}

end;

53 : begin {整数减,SUB}

t := t-1;

s[t].i := s[t].i-s[t+1].i {t-1栈顶，int存放两个整数-}

end;

54 : begin {实数加,ADR}

t := t-1;

s[t].r := s[t].r+s[t+1].r; {t-1栈顶，int存放两个实数+}

end;

55 : begin {实数减,SUR}

t := t-1;

s[t].r := s[t].r-s[t+1].r; {t-1栈顶，int存放两个实数-}

end;

56 : begin {与运算,AND}

t := t-1;

s[t].b := s[t].b and s[t+1].b {t-1栈顶，布尔值存放两个bool与}

end;

57 : begin {整数乘,MUL}

t := t-1;

s[t].i := s[t].i\*s[t+1].i {t-1栈顶，int存放两个整数\*}

end;

58 : begin {整数除法,DIV}

t := t-1;

if s[t+1].i = 0 {除0异常}

then ps := divchk

else s[t].i := s[t].i div s[t+1].i {t-1栈顶，int存放两个整数/}

end;

59 : begin {取模运算,MOD}

t := t-1;

if s[t+1].i = 0 {除0异常}

then ps := divchk

else s[t].i := s[t].i mod s[t+1].i {t-1栈顶，int存放两个整数mod}

end;

end { case }

end; { inter5 }

procedure inter6;

begin

case ir.f of

60 : begin {MUR，实数乘}

t := t-1;

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

end;

61 : begin {DIR，实数除}

t := t-1;

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

end;

62 : if eof(prd) {RDL，读完一行，如果到达尾部，异常，否则读一行}

then ps := redchk

else readln;

63 : begin {WRL，换行写}

writeln(prr); {写入prr行}

lncnt := lncnt+1; {行数+1}

chrcnt := 0; {从行首开始}

if lncnt > linelimit {如果行超出，异常}

then ps := linchk

end

end { case };

end; { inter6 }

begin { interpret } {下面对指令进行翻译，调用上述函数}

s[1].i := 0;

s[2].i := 0;

s[3].i := -1;

s[4].i := btab[1].last;

display[0] := 0;

display[1] := 0;

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

b := 0;

pc := tab[s[4].i].adr;

lncnt := 0;

ocnt := 0;

chrcnt := 0;

ps := run;

fld[1] := 10;

fld[2] := 22;

fld[3] := 10;

fld[4] := 1;

repeat

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 {如果结束}

then begin

writeln(prr); {换行写}

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

case ps of {根据不同错误信息进行报错}

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 ); {输出栈中信息}

blkcnt := blkcnt-1;

if blkcnt = 0

then h1 := 0;

h2 := s[h1+4].i;

if h1 <> 0

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

h2 := btab[tab[h2].ref].last;

while h2 <> 0 do

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

until h1 < 0

end;

writeln(prr);

writeln(prr,ocnt,' steps'); {一共执行了多少步}

end; { interpret }

procedure setup; {程序运行前的初始化准备}

begin

key[1] := 'and '; {定义保留字}

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; {定义保留字对应的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; {定义特殊字符定义的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）、函数和过程的名字，以及它们的相应信息}

begin

enter(' ',vvariable,notyp,0); { sentinel } {初始登录基本变量}

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); {初始登录基本函数}

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); {初始登录基本过程}

enter('readln ',prozedure,notyp,2);

enter('write ',prozedure,notyp,3);

enter('writeln ',prozedure,notyp,4);

enter(' ',prozedure,notyp,0);

end;

begin { main } {编译器执行}

setup; {初始化变量}

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

typebegsys := [ ident, arraysy, recordsy ]; {类型的开始符号集合}

blockbegsys := [ constsy, typesy, varsy, procsy, funcsy, beginsy ]; {分语句的开始符号集合}

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

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

stantyps := [ notyp, ints, reals, bools, chars ]; {基本变量的符号集合}

lc := 0; {重置pc}

ll := 0; {重置当前行的长度}

cc := 0; {重置当前行位置指针}

ch := ' '; {重置当前符号}

errpos := 0; {重置错误位置}

errs := []; {重置错误集合}

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

writeln;

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

readln( inf );

assign( psin, inf );

reset( psin );

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

readln( outf );

assign( psout, outf );

rewrite( psout );

assign ( prd, 'con' );

write( 'result file : ' ); {获取结果输出文件}

readln( fprr );

assign( prr, fprr );

reset ( prd );

rewrite( prr );

t := -1; {设置tab栈顶初值}

a := 0; {设置atab栈顶初值}

b := 1; {设置btab栈顶初始值，上来就是第一层}

sx := 0; {设置stab栈顶初值}

c2 := 0; {设置rconst栈顶初值}

display[0] := 1; {设置display初值}

iflag := false; {初始化一系列flag的值}

oflag := false;

skipflag := false;

prtables := false;

stackdump := false;

insymbol; {获得第一个sym}

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 {第二个单词，如果不是标识符报错，也就是不是程序名报错}

then error(2)

else begin

progname := id;

insymbol;

if sy <> lparent {如果不是左小括号报错}

then error(9)

else repeat {读取程序运行参数}

insymbol;

if sy <> ident

then error(2)

else begin

if id = 'input ' {可以没有输入文件}

then iflag := true

else if id = 'output ' {不能没有输出文件}

then oflag := true

else error(0);

insymbol

end

until sy <> comma; {直到sy不是逗号，说明读完}

if sy = rparent {文件路径作为参数之后，右括号结束}

then insymbol

else error(4);

if not oflag then error(20)

end

end;

enterids; {初始化变量}

with btab[1] do {载入主程序？}

begin

last := t;

lastpar := 1;

psize := 0;

vsize := 0;

end;

block( blockbegsys + statbegsys, false, 1 ); {分析处理主程序}

if sy <> period {pascal程序最后以.句号结尾，没有报错}

then error(2);

emit(31); { halt } {终止程序的P-code}

if prtables

then printtables;

if errs = []

then interpret {编译完后执行}

else begin {有错误的话提示}

writeln( psout );

writeln( psout, 'compiled with errors' );

writeln( psout );

errormsg;

end;

writeln( psout );

close( psout ); {关闭文件}

close( prr )

end.