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

}

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

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

llng = 121; { input line length }

emax = 322; { max exponent of real numbers }

emin = -292; { min exponent }

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

tmax = 100; { size of table }

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 }

lmax = 7; { maximum level }

smax = 600; { size of string-table }

ermax = 58; { max error no. }

omax = 63; { highest order code }

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

nmax = 32767; { 2\*\*15-1 }

lineleng = 132; { output line length }

linelimit = 200;

stacksize = 1450;

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);

index = -xmax..+xmax;

alfa = packed array[1..alng]of char; {\*alfa存储的是标识符，packed指连续存储\*}

objecttyp = (konstant, vvariable, typel, prozedure, funktion );

types = (notyp, ints, reals, bools, chars, arrays, records );

symset = set of symbol; {\*set表示子集\*}

typset = set of types;

item = record {\*pascal记录变量\*}

typ: types;

ref: index;

end;

order = packed record

f: -omax..+omax; {\*表示范围，数值参照之前的定义:63\*}

x: -lmax..+lmax;

y: -nmax..+nmax

end;

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;

t,a,b,sx,c1,c2:integer; { indices to tables } {\*指向表的索引/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 }

progname: alfa; {\*alfa是一种数据类型\*}

stantyps: typset;

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

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

key: array[1..nkw] of alfa; {\*key的元素是alfa类型\*}

ksy: array[1..nkw] of symbol;

sps: array[char]of symbol; {\*数组的索引是一个具体的字符\*}

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

tab: array[0..tmax] of { indentifier lable }

packed record {\*以记录变量定义符号表\*}

name: alfa; {\*每个变量有八项\*}

link: index;

obj: objecttyp;

typ: types;

ref: index;

normal: boolean;

lev: 0..lmax;

adr: integer

end;

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

packed record

inxtyp,eltyp: types;

elref,low,high,elsize,size: index

end;

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

packed record

last, lastpar, psize, vsize: index

end;

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

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

code: array[0..cmax] of order;{\*cmax定义了代码最大长度为800\*}

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; {\*注意到，所有的长度都是10，由alfa决定\*}

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); {\*写入输出文件psout\*}

writeln(psout,'key words');

k := 0;

while errs <> [] do {\*<>是不等于的pascal写法，这里是errs不为空\*}

begin

while not( k in errs )do k := k + 1; {\*找出所有出现的errmsg编号\*}

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

errs := errs - [k] {\*从errs中删去该项错误标号以继续进行while的判断\*}

end { while errs }

end { errormsg } ;

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

begin { underline skipped part of input }

while errpos < cc do

begin

write( psout, '-');

errpos := errpos + 1

end;

skipflag := false {\*skipflag：跳读的标志\*}

end { endskip };

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

begin

if cc = ll {\*字符计数=本行的字符长度，则\*}

then begin

if eof( psin ) {\*eof(t)返回值是boolean，如果读到psin输入的EOF\*}

then begin

writeln( psout );

writeln( psout, 'program incomplete' );

errormsg; {\*这里是调用procedure: errormsg\*}

exit; {\*注意这里退出了\*}

end;

if errpos <> 0

then begin

if skipflag then endskip; {\*如果需要跳，则调用procedure:endskip\*}

writeln( psout );

errpos := 0

end;

write( psout, lc: 5, ' '); {\*lc:program location counter，按5位格式化输出\*}

ll := 0;

cc := 0; {\*重置ll cc\*}

while not eoln( psin ) do {\*行末尾判断函数 返回boolean型\*}

begin

ll := ll + 1;

read( psin, ch );

write( psout, ch );

line[ll] := ch

end;

ll := ll + 1; {\*此处eoln返回true，读到行末尾\*}

readln( psin ); {\*处理行末尾符号(行结束符)\*}

line[ll] := ' ';

writeln( psout );

end;

cc := cc + 1;

ch := line[cc]; {\*再从line中取出下一个字符ch\*}

end { nextch };

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

begin

if errpos = 0

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

if cc > errpos

then begin

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

errpos := cc + 3;

errs := errs +[n]

end

end { error };

procedure fatal( n: integer ); {\*打印表格溢出的信息\*}

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

begin

writeln( psout );

errormsg; {\*先调用errormsg定位错误类型和信息\*}

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 };

procedure insymbol; {reads next symbol}{\*读取下一个单词符号，同时处理注释\*}

label 1,2,3; {\*为跳转指令准备的标签\*}

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

procedure readscale; {\*定义了一个处理指数+-的procedure\*}

var s,sign: integer;

begin

nextch;

sign := 1; {\*sign=1表示+，后面看到‘-’时sign=-1\*}

s := 0;

if ch = '+'

then nextch {\*读到+则再读一个\*}

else if ch = '-'

then begin

nextch;

sign := -1

end;

if not(( ch >= '0' )and (ch <= '9' )) {\*读完+-后面必须是数字\*}

then error( 40 )

else repeat

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

nextch;

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

e := s\*sign + e {\*e为最终的运算结果\*}

end { readscale };

procedure adjustscale;

var s : integer;

d, t : real;

begin

if k + e > emax {\*322，实型最大指数\*}

then error(21)

else if k + e < emin {\*-292，最小指数\*}

then rnum := 0 {\*直接忽略\*}

else begin

s := abs(e);

t := 1.0; {\*指数部分运算结果为t，这里赋值为1.0\*}

d := 10.0; {\*底数为10\*}

repeat

while not odd(s) do {\*s是奇数时返回true，这里将偶次幂处理为奇数次幂\*}

begin

s := s div 2;

d := sqr(d) {\*平方处理，因为指数除2所以平方\*}

end;

s := s - 1;

t := d \* t

until s = 0;

if e >= 0

then rnum := rnum \* t {\*之前取绝对值，在这里体现正负\*}

else rnum := rnum / t

end

end { adjustscale };

procedure options;

procedure switch( var b: boolean );

begin

b := ch = '+';

if not b

then if not( ch = '-' )

then begin { print error message }

while( ch <> '\*' ) and ( ch <> ',' ) do {\*跳过所有\*和，\*}

nextch; {\*nextch中是含有error处理的\*}

end

else nextch {\*当前为“-”\*}

else nextch {\*当前为“+”\*}

end { switch };

begin { options }

repeat

nextch;

if ch <> '\*' {\*编译选项写在注释中，所以先处理掉所有“\*”\*}

then begin

if ch = 't'

then begin

nextch;

switch( prtables ) {\*很早之前定义过的boolean型\*}

end

else if ch = 's'

then begin

nextch;

switch( stackdump )

end;

end

until ch <> ',' {\*“，”说明编译选项不止一个\*}

end { options };

begin { insymbol }

1: while( ch = ' ' ) or ( ch = chr(9) ) do {\*这里有一个跳转标签’1:’\*}

nextch; { space & htab } {\*跳过所有空格和\t\*}

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 } {\*首个字符是字母，表示是一个单词，开始识别\*}

k := 0;

id := ' '; {\*将id赋值为10个‘ ’\*}

repeat

if k < alng

then begin

k := k + 1;

id[k] := ch {\*将读取的ch写入id中，直到k>=alng或ch不为字母数字\*}

end;

nextch

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

i := 1;

j := nkw; { binary search }

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] {\*二分法查找，获取id对应的symbol\*}

else sy := ident {\*查找失败，当前不是保留字，是一个普通标识符\*}

end;

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

begin { number } {\*这里仍是switch的分支，若为数字\*}

k := 0;

inum := 0;

sy := intcon; {\*intcon是保留字类型，这里设置为intcon表示整数\*}

repeat

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

k := k + 1;

nextch {\*转化为整型存入inum\*}

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

if( k > kmax ) or ( inum > nmax ) {\*数字范围合法性检查\*}

then begin

error(21);

inum := 0;

k := 0 {\*出错记得重置\*}

end;

if ch = '.' {\*读到小数点，判断为小数部分\*}

then begin

nextch;

if ch = '.' {\*紧接着还是小数点，说明是表示一个范围\*}

then ch := ':'

else begin

sy := realcon; {\*将sy赋值为实型realcon\*}

rnum := inum;

e := 0; {\*处理小数部分\*}

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

begin

e := e - 1;

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

nextch

end;

if e = 0 {\*没读到小数部分，表示格式有误，报错\*}

then error(40);

if ch = 'e' {\*指数位的标记\*}

then readscale; {\*调用刚刚的readscale处理指数部分\*}

if e <> 0 then adjustscale {\*之前的procedure，计算结果\*}

end

end

else if ch = 'e' {\*无小数点小数部分，直接是指数部分，处理基本同上\*}

then begin

sy := realcon;

rnum := inum;

e := 0;

readscale;

if e <> 0

then adjustscale

end;

end;

':':

begin

nextch;

if ch = '=' {\*判断是冒号还是赋值语句\*}

then begin

sy := becomes;

nextch

end

else sy := colon {\*标记为一个冒号\*}

end;

'<': {\*可能是<= <> <三种\*}

begin

nextch;

if ch = '='

then begin

sy := leq;

nextch

end

else

if ch = '>'

then begin

sy := neq;

nextch

end

else sy := lss

end;

'>':

begin

nextch;

if ch = '='

then begin

sy := geq;

nextch

end

else sy := gtr

end;

'.': {\*两种可能，小数点.或者范围符号“..”\*}

begin

nextch;

if ch = '.'

then begin

sy := colon;

nextch

end

else sy := period

end;

'''': {\*四个单引号\*}

begin

k := 0;

2: nextch;

if ch = ''''

then begin

nextch;

if ch <> ''''

then goto 3

end;

if sx + k = smax {\*字符串表溢出\*}

then fatal(7);

stab[sx+k] := ch;

k := k + 1;

if cc = 1

then begin { end of line }

k := 0;

end

else goto 2; {\*用goto实现跳转和循环\*}

3: if k = 1 {\*分为charcon和stringcon两种\*}

then begin

sy := charcon;

inum := ord( stab[sx] )

end

else if k = 0 {\*空引号按报错处理\*}

then begin

error(38);

sy := charcon;

inum := 0

end

else begin

sy := stringcon;

inum := sx;

sleng := k;

sx := sx + k

end

end;

'(':

begin

nextch;

if ch <> '\*'

then sy := lparent {\*lparent左括号的标识symbol\*}

else begin { comment }

nextch;

if ch = '$'

then options; {\*编译选项的options procedure\*}

repeat

while ch <> '\*' do nextch;

nextch

until ch = ')';

nextch;

goto 1

end

end;

'{':

begin

nextch;

if ch = '$'

then options; {\*同上，需要判断是左大括号还是编译选项\*}

while ch <> '}' do

nextch;

nextch;

goto 1

end;

'+', '-', '\*', '/', ')', '=', ',', '[', ']', ';': {\*特殊符号一起处理\*}

begin

sy := sps[ch];

nextch

end;

'$','"' ,'@', '?', '&', '^', '!': {\*非法的开头字符，一起报错\*}

begin

error(24);

nextch;

goto 1

end

end { case }

end { insymbol };

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

{\*x0标识符名 x1标识符种类 x2标识符类型\*}

begin {\*把标准类型、过程、函数名登录到符号表tab中\*}

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;

adr := x3; {\*int型的x3其实是一个地址/大小\*}

end

end; { enter }

procedure enterarray( tp: types; l,h: integer ); {\*将数组信息填入数组信息向量表\*}

begin

if l > h {\*应该是low high的意思，若不合理则报错\*}

then error(27);

if( abs(l) > xmax ) or ( abs(h) > xmax ) {\*超出最大限制\*}

then begin

error(27);

l := 0;

h := 0;

end;

if a = amax {\*溢出\*}

then fatal(4)

else begin

a := a + 1;

with atab[a] do

begin

inxtyp := tp;

low := l;

high := h

end

end

end { enterarray };

procedure enterblock; {\*向分程序表btab登记信息\*}

begin

if b = bmax

then fatal(2) {\*除了溢出不用担心其他类型的出错\*}

else begin

b := b + 1;

btab[b].last := 0; {\*last为指向该分程序最后一个符号在表中的位置\*}

btab[b].lastpar := 0; {\* lastpar为分程序最后一个参数符号在表中的位置\*}

end

end { enterblock };

procedure enterreal( x: real ); {\*登记实常数表，传入x\*}

begin

if c2 = c2max – 1 {\*c2max=20，根据后文，19即为溢出\*}

then fatal(3)

else begin

rconst[c2+1] := x; {\*将x写入\*}

c1 := 1; {\*c1表示的是第一个x的值的位置，即之前是否有同值元素\*}

while rconst[c1] <> x do

c1 := c1 + 1;

if c1 > c2

then c2 := c1

end

end { enterreal };

procedure emit( fct: integer ); {\*这三个都是生成p-code的指令\*}

begin

if lc = cmax

then fatal(6);

code[lc].f := fct; {\*fct是操作码\*}

lc := lc + 1

end { emit };

procedure emit1( fct, b: integer );

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 );

begin

if lc = cmax then fatal(6);

with code[lc] do

begin

f := fct;

x := a;

y := b

end;

lc := lc + 1;

end { emit2 };

procedure printtables; {\*打印，包括：符号表 btab 实常数表 p-code\*}

var i: integer;

o: order;

mne: array[0..omax] of {\*omax=63\*}

packed array[1..5] of char; {\*每个长5位的连续地址空间\*}

begin

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

writeln(psout);

writeln(psout);

writeln(psout);

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

writeln(psout);

for i := btab[1].last to t do {\*btab[1]最后一个标识符到当前标识符之间所有信息\*}

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);

{\*ord=ordinal，返回的是顺序的数值\*}

writeln( psout );

writeln( psout );

writeln( psout );

writeln( psout, 'blocks last lpar psze vsze' ); {\*分程序表btab的内容\*}

writeln( psout );

for i := 1 to b do

with btab[i] do

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

writeln( psout );

writeln( psout );

writeln( psout );

writeln( psout, 'arrays xtyp etyp eref low high elsz size'); {\*atab中的信息\*}

writeln( psout );

for i := 1 to a do

with atab[i] do

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

writeln( psout );

writeln( psout );

writeln( psout );

writeln( psout, 'code:'); {\*PCODE部分\*}

writeln( psout );

for i := 0 to lc-1 do {\*lc是program location counter，为当前pcode最大位置\*}

begin

write( psout, i:5 );

o := code[i]; {\*code[i]是一个order，含有f,x,y三个属性\*}

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 )

end { printtables };

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

type conrec = record

case tp: types of

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

reals :( r:real )

end; {\*swtich tp的类型来决定conrec的内容是哪一种\*}

var dx : integer ; { data allocation index }

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

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

x : integer ;

procedure skip( fsys:symset; n:integer); {\*用来跳过出错的block\*}

begin

error(n);

skipflag := true; {\*这里对skipflag做了改动，联系之前的endskip\*}

while not ( sy in fsys ) do

insymbol; {\*直到读入一个setsym中的symbol\*}

if skipflag then endskip

end { skip };

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

begin

if not( sy in s1 )

then skip( s1 + s2, n ) {\*检验sym是否合法，并skip不合法的内容\*}

end { test };

procedure testsemicolon; {\*检验当前符号是否为分号\*}

begin

if sy = semicolon {\*之间标记的semicolon sym\*}

then insymbol

else begin

error(14);

if sy in [comma, colon]

then insymbol

end;

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

end { testsemicolon };

procedure enter( id: alfa; k:objecttyp ); {\*区别于上一个enter，这个是分程序内的\*}

var j,l : integer;

begin

if t = tmax {\*已满，溢出\*}

then fatal(1)

else begin

tab[0].name := id;

j := btab[display[level]].last; {\*分程序内，所以有level标识，取出相应分程序最后一个标识符的位置\*}

l := j;

while tab[j].name <> id do

j := tab[j].link;

if j <> 0

then error(1)

else begin

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id在符号表中的位置\*}

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

begin

i := level;

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

repeat

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

while tab[j].name <> id do

j := tab[j].link;

i := i - 1;

until ( i < 0 ) or ( j <> 0 );

if j = 0

then error(0);

loc := j

end { loc } ;

procedure entervariable; {\*将变量名登记到符号表中\*}

begin

if sy = ident

then begin

enter( id, vvariable );

insymbol

end

else error(2)

end { entervariable };

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

{\*处理出现的常量，并返回c(常量的类型和数值)\*}

var x, sign : integer;

begin

c.tp := notyp; {\*c也是一个标记变量，分别是类型和值\*}

c.i := 0;

test( constbegsys, fsys, 50 ); {\*先检验合法性\*}

if sy in constbegsys {\*这个是在后面定义的，暂不着急\*}

then begin

if sy = charcon {\*是字符常量\*}

then begin

c.tp := chars;

c.i := inum;

insymbol

end

else begin

sign := 1; {\*默认是“+”值为1\*}

if sy in [plus, minus] {\*其实就是在判断sy是不是“+ -”\*}

then begin

if sy = minus

then sign := -1;

insymbol {\*这里再读一个symbol\*}

end;

if sy = ident {\*判断标识符常量\*}

then begin

x := loc(id); {\*procedure loc找到id在符号表的位置\*}

if x <> 0

then

if tab[x].obj <> constant {\*找出来的符号不是常量，error\*}

then error(25)

else begin

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

if c.tp = reals

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

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

end;

insymbol

end

else if sy = intcon {\*是整形常量\*}

then begin

c.tp := ints;

c.i := sign\*inum; {\*把刚才处理的符号sign存进去\*}

insymbol

end

else if sy = realcon

then begin

c.tp := reals;

c.r := sign\*rnum;

insymbol

end

else skip(fsys,50) {\*不是以上任何一种就是出错了，skip\*}

end;

test(fsys,[],6)

end

end { constant };

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

{\*处理类型描述，由参数得到类型tp，指向类型详细信息表的指针ref，类型的大小sz\*}

var eltp : types;

elrf, x : integer;

elsz, offset, t0, t1 : integer;

procedure arraytyp( var aref, arsz: integer );{\*这里少了一个缩进？arraytyp应该是子程序吧\*}

var eltp : types; {\*数组类型比较特殊，单独处理\*}

low, high : conrec;

elrf, elsz: integer;

begin

constant( [colon, rbrack, rparent, ofsy] + fsys, low ); {\*返回类型和大小\*}

if low.tp = reals {\*指下标竟然不是整数，报错\*}

then begin

error(27);

low.tp := ints;

low.i := 0

end;

if sy = colon {\*描述数组时，下一个是：可以继续执行\*}

then insymbol

else error(13);

constant( [rbrack, comma, rparent, ofsy ] + fsys, high );

if high.tp <> low.tp

then begin

error(27);

high.i := low.i

end;

enterarray( low.tp, low.i, high.i ); {\*将数组的下标类型、值，上标值填入atab\*}

aref := a; {\*将数组的地址赋值给aref\*}

if sy = comma {\*读到，意味着这个数组是高维的，这也是数组特殊的地方\*}

then begin

insymbol;

eltp := arrays;

arraytyp( elrf, elsz ) {\*递归处理更高维数的数组\*}

end

else begin

if sy = rbrack {\*rbrack=右括号，表示数组下标部分声明完毕\*}

then insymbol

else begin

error(12);

if sy = rparent {\*读到右括号则继续insymbol，应该是容错的意思\*}

then insymbol

end;

if sy = ofsy {\*声明必须有of，没有则报错\*}

then insymbol

else error(8);

typ( fsys, eltp, elrf, elsz ) {\*用来处理元素的类型\*}

end;

with atab[aref] do {\*处理完毕将一维或多维数组的信息写入atab(aref)\*}

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 );

if sy in typebegsys

then begin

if sy = ident

then begin

x := loc(id); {\*继续寻找id在符号表中的位置\*}

if x <> 0

then with tab[x] do

if obj <> typel

then error(29)

else begin

tp := typ;

rf := ref;

sz := adr;

if tp = notyp {\*notyp指为定义类型，报错\*}

then error(30)

end;

insymbol

end

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

then begin

insymbol;

if sy = lbrack {\*读到数组的“[”符号，继续处理\*}

then insymbol

else begin

error(11);

if sy = lparent {\*这是一个容错处理\*}

then insymbol

end;

tp := arrays; {\*将tp的值/类型设置为array\*}

arraytyp(rf,sz)

end

else begin { records } {\*排除法，除了标识符和数组只可能是记录变量\*}

insymbol;

enterblock; {\*需要登记在分程序表中\*}

tp := records;

rf := b; {\*这里rf指的是该记录变量在btab中的位置\*}

if level = lmax {\*分程序层次/嵌套层次达到最大\*}

then fatal(5); {\*注意这里需要报错一个溢出\*}

level := level + 1;

display[level] := b; {\*建立分程序索引，display表格\*}

offset := 0;

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

begin { field section }

{\*底下都是在处理记录变量内变量的声明\*}

if sy = ident

then begin

t0 := t;

entervariable;

while sy = comma do

begin

insymbol;

entervariable

end;

if sy = colon

then insymbol

else error(5);

t1 := t;

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

while t0 < t1 do

begin

t0 := t0 + 1;

with tab[t0] do

begin

typ := eltp;

ref := elrf;

normal := true;

adr := offset;

offset := offset + elsz

end

end

end; { sy = ident }

if sy <> endsy

then begin

if sy = semicolon

then insymbol

else begin

error(14);

if sy = comma

then insymbol

end;

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

end

end; { field section }

btab[rf].vsize := offset;

sz := offset;

btab[rf].psize := 0;

insymbol;

level := level - 1

end; { record }

test( fsys, [],6 )

end;

end { typ };

procedure parameterlist; {\*处理过程或函数说明中的形式参数表\*}

{\*将形参的信息登记到符号表中\*}

var tp : types;

valpar : boolean;

rf, sz, x, t0 : integer;

begin

insymbol;

tp := notyp; {\*先设置为空类型和0\*}

rf := 0;

sz := 0;

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

while sy in [ident, varsy] do {\*处理形参中是标识符的或是var定义的\*}

begin

if sy <> varsy

then valpar := true

else begin

insymbol;

valpar := false

end;

t0 := t; {\*将tab中第一个参数的地址存入t0\*}

entervariable; {\*将变量名登记到符号表中\*}

while sy = comma do {\*“，”都好说明有多个同类型的变量，处理\*}

begin

insymbol;

entervariable;

end;

{\*这里的处理跟typ处理记录同类成员变量的代码相似\*}

if sy = colon

then begin

insymbol;

if sy <> ident

then error(2)

else begin

x := loc(id);

insymbol;

if x <> 0

then with tab[x] do

if obj <> typel

then error(29)

else begin

tp := typ;

rf := ref;

if valpar

then sz := adr

else sz := 1

end;

end;

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

end

else error(5); {\*接上，不是冒号需要报错\*}

while t0 < t do

begin

t0 := t0 + 1;

with tab[t0] do

begin

typ := tp;

ref := rf;

adr := dx; {\*adr的值为运行栈中分配单元的相对地址dx\*}

lev := level;

normal := valpar;

dx := dx + sz

end

end;

if sy <> rparent {\*不是右括号有两种情况，还有参数，或出错\*}

then begin

if sy = semicolon {\*没有semicolon则是出错error\*}

then insymbol

else begin

error(14);

if sy = comma

then insymbol

end;

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

end

end { while };

if sy = rparent {\*声明完参数，以右括号结尾\*}

then begin

insymbol;

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

end

else error(4)

end { parameterlist };

procedure constdec; {\*处理常量声明，将变量名和相应信息填入符号表\*}

var c : conrec;

begin

insymbol;

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

while sy = ident do

begin

enter(id, konstant); {\*将id以常量形式写入符号表\*}

insymbol;

if sy = eql {\*如果是等号则确定是合法的赋值\*}

then insymbol

else begin {\*常量声明不用等号，报错\*}

error(16);

if sy = becomes

then insymbol

end;

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

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

tab[t].ref := 0; {\*常量的ref域是0\*}

if c.tp = reals {\*如果常量是实型的\*}

then begin

enterreal(c.r);

tab[t].adr := c1; {\*adr域填入的是常量在rconst中的地址\*}

end

else tab[t].adr := c.i;

testsemicolon

end

end { constdec };

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

var tp: types;

rf, sz, t1 : integer;

begin

insymbol;

test([ident], blockbegsys,2 );

while sy = ident do

begin

enter(id, typel);

t1 := t;

insymbol;

if sy = eql

then insymbol

else begin

error(16);

if sy = becomes

then insymbol

end;

typ( [semicolon,comma,ident]+fsys, tp,rf,sz );

with tab[t1] do

begin

typ := tp;

ref := rf;

adr := sz

end;

testsemicolon

end

end { typedeclaration };

procedure variabledeclaration; {\*处理变量声明，应该是将变量声明也填入tab\*}

var tp : types;

t0, t1, rf, sz : integer;

begin

insymbol;

while sy = ident do

begin

t0 := t;

entervariable; {\*这里的操作基本与之前的相同\*}

while sy = comma do

begin

insymbol;

entervariable;

end;

if sy = colon

then insymbol

else error(5);

t1 := t;

typ([semicolon,comma,ident]+fsys, tp,rf,sz );

while t0 < t1 do

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 };

procedure procdeclaration; {\*处理过程和函数的声明，并填入btab\*}

var isfun : boolean;

begin

isfun := sy = funcsy; {\*funcsy表示方法，否则是过程procedure\*}

insymbol;

if sy <> ident {\*不是标识符则报错\*}

then begin

error(2);

id :=' ' {\*用10个空格代替\*}

end;

if isfun

then enter(id,funktion)

else enter(id,prozedure);

tab[t].normal := true;

insymbol;

block([semicolon]+fsys, isfun, level+1 );

if sy = semicolon

then insymbol

else error(14);

emit(32+ord(isfun)) {\*ord将isfun转成相应整型，emit是之前定义的生成PCODE procedure\*}

end { proceduredeclaration };

procedure statement( fsys:symset ); {\*处理完声明语句，接下来处理何种statement\*}

var i : integer;

{\*用于处理expression子程序，这里先声明一下(pascal的预定义机制)\*}

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

{\*用于处理结构变量，处理对象是v\*}

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

var x : item;

a,j : integer;

begin { sy in [lparent, lbrack, period] }

repeat

if sy = period

then begin

insymbol; { field selector } {\*这里表示读入symbol进入域\*}

if sy <> ident {\*域的类型必须为标识符\*}

then error(2)

else begin

if v.typ <> records {\*v的type必须为标记变量record\*}

then error(31)

else begin { search field identifier }

j := btab[v.ref].last; {\*该记录在分程序表中的最后一个标识符的地址\*}

tab[0].name := id;

while tab[j].name <> id do {\*从符号表中该位置开始查找id\*}

j := tab[j].link;

if j = 0 {\*因为是向前查找，j=0说明没有找到\*}

then error(0);

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

v.ref := tab[j].ref; {\*v.ref为当前域在btab中的位置\*}

a := tab[j].adr;

if a <> 0

then emit1(9,a) {\*定义的emit1生成PCODE\*}

{\*将a:integer放在栈顶\*}

end;

insymbol

end

end

else begin { array selector }

if sy <> lbrack {\*只认可“[”\*}

then error(11);

repeat

insymbol;

expression( fsys+[comma,rbrack],x); {\*递归调用，处理深层嵌套\*}

if v.typ <> arrays

then error(28)

else begin

a := v.ref;

if atab[a].inxtyp <> x.typ

then error(26)

else if atab[a].elsize = 1 {\*这里有疑惑\*}

then emit1(20,a)

else emit1(21,a);

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

v.ref := atab[a].elref

end

until sy <> comma;

if sy = rbrack {\*扫描到右括号“]”，正常结束\*}

then insymbol

else begin

error(12);

if sy = rparent

then insymbol

end

end

until not( sy in[lbrack, lparent, period]); {\*repeat的终止条件，不是“([.”\*}

test( fsys,[],6)

end { selector };

procedure call( fsys: symset; i:integer ); {\*处理非标准的函数过程/函数调用\*}

var x : item;

lastp,cp,k : integer;

begin

emit1(18,i); {\*这里生成的PCODE是在mark stack 生成标记栈\*}

lastp := btab[tab[i].ref].lastpar;

{\*当前过程函数最后一个参数在符号表中的位置\*}

cp := i; {\*这里cp是被调用的过程在符号表中的位置\*}

if sy = lparent

then begin { actual parameter list }

repeat

insymbol;

if cp >= lastp {\*过程地址>=最后一个参数位置，报错\*}

then error(39)

else begin {\*说明还有参数需要处理\*}

cp := cp + 1;

if tab[cp].normal {\*normal=true表示传入的是形参\*}

then begin { value parameter }

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

{\*这里调用exprssion处理参数\*}

if x.typ = tab[cp].typ

then begin

if x.ref <> tab[cp].ref

{\*与btab中地址不同，显然是出现了错误\*}

then error(36)

else if x.typ = arrays {\*数组型\*}

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

else if x.typ = records

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

end

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

{\*表达式的type是int，而分程序要求输入类型为real，需要强制类型转换\*}

then emit1(26,0)

else if x.typ <> notyp

then error(36);

end

else begin { variable parameter }

if sy <> ident

then error(2)

else begin

k := loc(id);

insymbol;

if k <> 0

then begin

if tab[k].obj <> vvariable

then error(37);

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

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

if tab[k].normal {\*值形参\*}

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

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

{\*变量形参\*}

if sy in [lbrack, lparent, period]

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;

if sy = rparent

then insymbol

else error(4)

end;

if cp < lastp {\*处理完还没到lastp，实参数不够，报错\*}

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

emit1(19,btab[tab[i].ref].psize-1 );

if tab[i].lev < level

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

end { call };

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

{\*处理整型或实数型两个操作数时的强制类型转换\*}

begin

if ( a > reals ) or ( b > reals ) {\*“>real”表示既不是real也不是int\*}

then begin

error(33);

resulttype := notyp

end

else if ( a = notyp ) or ( b = notyp ) {\*ab都是未定义类型，返回notyp\*}

then resulttype := notyp {\*不报错？\*}

else if a = ints

then if b = ints

then resulttype := ints

else begin

resulttype := reals; {\*result类型和b一致\*}

emit1(26,1)

end

else begin {\*a是real，返回值result必是real\*}

resulttype := reals;

if b = ints

then emit1(26,0)

end

end { resulttype } ;

{\*分析处理表达式，由参数x返回求值结果的类型\*}

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

var y : item;

op : symbol;

{\*分析处理简单表达式，由参数x返回求值结果的类型\*}

procedure simpleexpression( fsys: symset; var x: item );

var y : item;

op : symbol;

{\*处理项，由参数返回结果类型\*}

procedure term( fsys: symset; var x: item );

var y : item;

op : symbol;

{\*处理因子，由参数返回结果类型\*}

procedure factor( fsys: symset; var x: item );

var i,f : integer;

{\*处理标准函数调用\*}

procedure standfct( n: integer );

var ts : typset;

begin { standard function no. n }

if sy = lparent {\*标准函数调用必须由“(”开始\*}

then insymbol

else error(9);

if n < 17 {\*只处理编号17以内的标准函数\*}

then begin

expression( fsys+[rparent], x );

case n of

{ abs, sqr } 0,2: begin {\*绝对值与开方，只能有实型和整型\*}

ts := [ints, reals];

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

if x.typ = reals {\*实型则函数标号+1\*}

then n := n + 1

end;

{ odd, chr } 4,5: ts := [ints]; {\*判断奇偶性、ascii转字符串，只允许整型\*}

{ odr } 6: ts := [ints,bools,chars]; {\*odr只支持这三种\*}

{ succ,pred } 7,8 : begin {\*得到前后一个元素，支持这三种\*}

ts := [ints, bools,chars];

tab[i].typ := x.typ

end;

{ round,trunc } 9,10,11,12,13,14,15,16:

{ sin,cos,... } begin

ts := [ints,reals];

if x.typ = ints {\*整型需要转为实型，并生成PCODE\*}

then emit1(26,0)

end;

end; { case }

if x.typ in ts

then emit1(8,n)

else if x.typ <> notyp

then error(48);

end

else begin { n in [17,18] } {\*处理编号为17 18的标准函数\*}

if sy <> ident

then error(2)

else if id <> 'input ' {\*只有input时是合法的\*}

then error(0)

else insymbol;

emit1(8,n); {\*生成相应的标准函数\*}

end;

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

if sy = rparent {\*遇到右括号则结束\*}

then insymbol

else error(4)

end { standfct } ; {\*标准函数处理结束\*}

begin { factor } {\*因子分析程序开始\*}

x.typ := notyp;

x.ref := 0; {\*初始设定同上\*}

test( facbegsys, fsys,58 ); {\*检验合法性\*}

while sy in facbegsys do {\*从所有合法的facbegsys因子开始符号循环处理\*}

begin

if sy = ident {\*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，生成PCODE\*}

x.typ := typ;

x.ref := 0;

if x.typ = reals

then emit1(25,adr)

{\*将实数入栈，adr对应其在rconst中的位置\*}

else emit1(24,adr)

end;

vvariable:begin {\*这里处理变量\*}

x.typ := typ;

x.ref := ref;

if sy in [lbrack, lparent,period]

{\*表示是数组变量或记录变量，否则是一个普通变量\*}

then begin

if normal

{\*normal: 变量形参为false，值参和其他为为true，标准名的normal域也为true\*}

then f := 0

else f := 1;

emit2(f,lev,adr);

selector(fsys,x);

if x.typ in stantyps

then emit(34)

end

else begin {\*处理普通变量\*}

if x.typ in stantyps

then if normal

then f := 1

else f := 2

else if normal

then f := 0

else f := 1;

emit2(f,lev,adr)

end

end;

typel,prozedure: error(44); {\*因子不能使类型名或过程\*}

funktion: begin

x.typ := typ;

if lev <> 0 {\*lev不为0表示非标准函数\*}

then call(fsys,i)

else standfct(adr)

end

end { case,with }

end

else if sy in [ charcon,intcon,realcon ] {\*表示sy是一个常量\*}

then begin

if sy = realcon {\*分三类处理\*}

then begin

x.typ := reals;

enterreal(rnum); {\*登记实常数表\*}

emit1(25,c1)

end

else begin

if sy = charcon

then x.typ := chars

else x.typ := ints;

emit1(24,inum)

end;

x.ref := 0;

insymbol

end

else if sy = lparent {\*左括号，说明接下来是一个表达式\*}

then begin

insymbol;

expression(fsys + [rparent],x);

{\*分析处理表达式，x为返回值\*}

if sy = rparent

then insymbol

else error(4)

end

else if sy = notsy {\*未定义/非关键字\*}

then begin

insymbol;

factor(fsys,x); {\*调用factor处理\*}

if x.typ = bools

then emit(35)

else if x.typ <> notyp

then error(32)

end;

test(fsys,facbegsys,6)

end { while }

end { factor };

begin { term } {\*处理项 term\*}

factor( fsys + [times,rdiv,idiv,imod,andsy],x); {\*调用factor处理[]中的因子\*}

while sy in [times,rdiv,idiv,imod,andsy] do

begin

op := sy;

insymbol;

factor(fsys+[times,rdiv,idiv,imod,andsy],y ); {\*y为二元操作的第二个操作数\*}

if op = times {\*times表示乘法\*}

then begin

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

case x.typ of

notyp: ;

ints : emit(57); {\*整型 实型的乘法调用emit\*}

reals: emit(60);

end

end

else if op = rdiv {\*实型的除法 real-divided\*}

then begin

if x.typ = ints

then begin {\*x为int则进行强制转换\*}

emit1(26,1);

x.typ := reals;

end;

if y.typ = ints {\*同理强制转换y\*}

then begin

emit1(26,0);

y.typ := reals;

end;

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

then emit(61) {\*最后检查一遍开始除\*}

else begin {\*不要忘了考虑notyp的情况\*}

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

then error(33);

x.typ := notyp

end

end

else if op = andsy {\*and-symbol 与操作的意思\*}

then begin

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

then emit(56)

else begin {\*运算这里永远不要忘了考虑notyp\*}

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

then error(32);

x.typ := notyp

end

end

else begin { op in [idiv,imod] }

{\*idiv整形除，imod取模(整形操作)\*}

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

then if op = idiv

then emit(58)

else emit(59)

else begin {\*处理notyp\*}

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

then error(34);

x.typ := notyp

end

end

end { while }

end { term };

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

if sy in [plus,minus] {\*先处理符号\*}

then begin

op := sy;

insymbol;

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

if x.typ > reals {\*>reals即不是int或real型，报错\*}

then error(33)

else if op = minus

then emit(36)

end

else term(fsys+[plus,minus,orsy],x); {\*处理循环出现的符号\*}

while sy in [plus,minus,orsy] do

begin

op := sy;

insymbol;

term(fsys+[plus,minus,orsy],y); {\*y为处理后的项\*}

if op = orsy {\*or-symbol，处理“or”\*}

then begin

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

then emit(51)

else begin

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

then error(32);

x.typ := notyp

end

end

else begin

x.typ := resulttype(x.typ,y.typ); {\*将运算后的reault-type赋值给x\*}

case x.typ of

notyp: ;

ints: if op = plus

then emit(52)

else emit(53);

reals:if op = plus

then emit(54)

else emit(55)

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]

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 {\*这里是数值比较符号\*}

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

neq: emit(46); {\*!= <>\*}

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

leq: emit(48); {\*<=\*}

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

geq: emit(50); {\*?=\*}

end

else begin {\*比较时，两个运算数类型不一致则做转换\*}

if x.typ = ints

then begin

x.typ := reals;

emit1(26,1)

end

else if y.typ = ints

then begin

y.typ := reals;

emit1(26,0)

end;

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

then case op of

eql: emit(39); {\*操作符的意义和处理同上\*}

neq: emit(40);

lss: emit(41);

leq: emit(42);

gtr: emit(43);

geq: emit(44);

end

else error(35)

end;

x.typ := bools

end

end { expression };

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

var x,y: item;

f : integer;

begin { tab[i].obj in [variable,prozedure] }

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] {\*左括号，表示赋值的变量是一个数组\*}

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

if sy = becomes

then insymbol

else begin

error(51);

if sy = eql {\*对于赋值操作，“=”可以容错\*}

then insymbol

end;

expression(fsys,y);

if x.typ = y.typ {\*情况一，赋值两边类型一致\*}

then if x.typ in stantyps

then emit(38)

else if x.ref <> y.ref

then error(46)

else if x.typ = arrays

then emit1(23,atab[x.ref].size)

else emit1(23,btab[x.ref].vsize)

else if(x.typ = reals )and (y.typ = ints) {\*情况二，设计强制转换\*}

then begin

emit1(26,0);

emit(38)

end

else if ( x.typ <> notyp ) and ( y.typ <> notyp ) {\*情况三，notyp\*}

then error(46)

end { assignment };

procedure compoundstatement; {\*处理带有begin和end的复合语句\*}

begin

insymbol;

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

while sy in [semicolon]+statbegsys do {\*如果紧接着读到分号或者statbegsys\*}

begin {\*则开始处理复合语句\*}

if sy = semicolon

then insymbol

else error(14);

statement([semicolon,endsy]+fsys)

end;

if sy = endsy {\*读到end表示该语句正常结束\*}

then insymbol

else error(57) {\*否则不匹配，报错\*}

end { compoundstatement };

procedure ifstatement; {\*处理if语句\*}

var x : item;

lc1,lc2: integer;

begin

insymbol;

expression( fsys+[thensy,dosy],x); {\*将从if到then/do之间的语句交给expression procedure\*}

if not ( x.typ in [bools,notyp]) {\*if的内容x的typ必须是real，否则报错\*}

then error(17);

lc1 := lc;

emit(11); { jmpc }

if sy = thensy

then insymbol

else begin

error(52); {\*报错是因为没有读到then，但是这里容错了继续处理do\*}

if sy = dosy

then insymbol

end;

statement( fsys+[elsesy]); {\*将else分支之前的代码交给statement处理\*}

if sy = elsesy

then begin

insymbol;

lc2 := lc;

emit(10);

code[lc1].y := lc;

statement(fsys);

code[lc2].y := lc

end

else code[lc1].y := lc

end { ifstatement };

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

var x : item;

i,j,k,lc1 : integer;

casetab : array[1..csmax]of {\*case的分支个数有限制：csmax=30\*}

packed record {\*用packed的方式连续存储\*}

val,lc : index

end;

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

procedure caselabel; {\*caselabel指处理case中的标号，将对应的目标代码填入casetab\*}

var lab : conrec;

k : integer;

begin

constant( fsys+[comma,colon],lab ); {\*将冒号之前的常量交给constant处理\*}

if lab.tp <> x.typ

then error(47)

else if i = csmax

then fatal(6) {\*casetab也需要考虑溢出\*}

else begin

i := i+1; {\*移动case 在表中的指针\*}

k := 0; {\*用来检查是否重复定义\*}

casetab[i].val := lab.i;

casetab[i].lc := lc; {\*分别是case的value和location\*}

repeat

k := k+1

until casetab[k].val = lab.i;

if k < I {\*重复声明\*}

then error(1); { multiple definition }

end

end { caselabel };

procedure onecase; {\*处理case的一个分支\*}

begin

if sy in constbegsys {\*case的情况必须是常量\*}

then begin

caselabel; {\*调用之前获得一个标签\*}

while sy = comma do {\*！！注意，一个case可能对应多个标签\*}

begin

insymbol;

caselabel

end;

if sy = colon {\*读到冒号才是真正结束\*}

then insymbol

else error(5);

statement([semicolon,endsy]+fsys);

j := j+1;

exittab[j] := lc; {\*这里记录的是case分支结束的代码地址\*}

emit(10) {\*之后生成跳转指令要用到\*}

end

end { onecase };

begin { casestatement } {\*开始处理case的statement部分\*}

insymbol;

i := 0;

j := 0;

expression( fsys + [ofsy,comma,colon],x );

if not( x.typ in [ints,bools,chars,notyp ]) {\*老样子，先处理声明，以of为标志\*}

then error(23); {\*不是以上四种则报错\*}

lc1 := lc;

emit(12); {jmpx}

if sy = ofsy

then insymbol

else error(8);

onecase; {\*先处理一个分支（因为至少一个）\*}

while sy = semicolon do {\*循环处理其余所有分支，直到读到semicolon\*}

begin

insymbol;

onecase

end;

code[lc1].y := lc; {\*存地址\*}

for k := 1 to i do {\*遍历所有分支建立casetab\*}

begin

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

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

end;

emit1(10,0);

for k := 1 to j do

code[exittab[k]].y := lc;

if sy = endsy {\*读到end时结束\*}

then insymbol

else error(57)

end { casestatement };

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

var x : item;

lc1: integer;

begin

lc1 := lc; {\*lcl保存repeat刚开始时的locatin\*}

insymbol;

statement( [semicolon,untilsy]+fsys); {\*循环体中的语句交给statement处理\*}

while sy in [semicolon]+statbegsys do

begin {\*如果遇到了seimicolon分号或者state的语句，说明还有语句\*}

if sy = semicolon {\*在这个循环中继续处理\*}

then insymbol

else error(14);

statement([semicolon,untilsy]+fsys)

end;

if sy = untilsy {\*直至读到until语句\*}

then begin

insymbol;

expression(fsys,x);

if not(x.typ in [bools,notyp] ) {\*until是一个判断语句，类型必须为boolean\*}

then error(17);

emit1(11,lc1);

end

else error(53)

end { repeatstatement };

procedure whilestatement; {\*处理while语句\*}

var x : item;

lc1,lc2 : integer;

begin

insymbol;

lc1 := lc;

expression( fsys+[dosy],x); {\*while到do之间的内容，返回x\*}

if not( x.typ in [bools, notyp] ) {\*显然x必须是boolean\*}

then error(17);

lc2 := lc; {\*同上，需要记录循环开始的地址lc\*}

emit(11);

if sy = dosy {\*do之后的内容做普通处理\*}

then insymbol

else error(54);

statement(fsys);

emit1(10,lc1); {\*似乎是跳转指令PCODE\*}

code[lc2].y := lc

end { whilestatement };

procedure forstatement; {\*处理for语句\*}

var cvt : types;

x : item;

i,f,lc1,lc2 : integer;

begin

insymbol;

if sy = ident {\*for循环体的开头是标识符\*}

then begin

i := loc(id); {\*查找id在符号表中的位置\*}

insymbol;

if i = 0 {\*说明没找到\*}

then cvt := ints {\*默认为int整型\*}

else if tab[i].obj = vvariable

then begin

cvt := tab[i].typ;

if not tab[i].normal {\*不是实参，是变量形参，报错\*}

then error(37)

else emit2(0,tab[i].lev, tab[i].adr );

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

then error(18)

end

else begin {\*符号也不是个变量，报错，将cvt(计数变量)设为integer整型\*}

error(37);

cvt := ints

end

end

else skip([becomes,tosy,downtosy,dosy]+fsys,2);

{\*for的开头没有标识符，出错，直接skip跳过直至读到接下来的赋值符号\*}

if sy = becomes

then begin

insymbol;

expression( [tosy, downtosy,dosy]+fsys,x);

if x.typ <> cvt

then error(19);

end

else skip([tosy, downtosy,dosy]+fsys,51);

f := 14;

if sy in [tosy,downtosy] {\*“to downto”说明需要设定步长\*}

then begin

if sy = downtosy {\*downto，按步长减少\*}

then f := 16; {\*将f的值设定为16\*}

insymbol;

expression([dosy]+fsys,x); {\*处理，直到读到do\*}

if x.typ <> cvt

then error(19)

end

else skip([dosy]+fsys,55); {\*if语句一定要有todownto，不然跳过\*}

lc1 := lc; {\*记录循环位置，这里是F1U指令的位置\*}

emit(f); {\*根据f的值生成PCODE\*}

if sy = dosy {\*开始处理do的内容部分\*}

then insymbol

else error(54);

lc2 := lc; {\*lc2是循环开始的代码位置\*}

statement(fsys);

emit1(f+1,lc2);

code[lc1].y := lc

end { forstatement };

procedure standproc( n: integer ); {\*标准过程 standard procedure\*}

var i,f : integer;

x,y : item;

begin

case n of

1,2 : begin { read } {\*标准读入\*}

if not iflag

then begin

error(20);

iflag := true

end;

if sy = lparent {\*还是需要先读入一个左括号\*}

then begin

repeat

insymbol;

if sy <> ident {\*读入的参数不是标识符报错\*}

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

then f := 0

else f := 1;

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))

else error(41)

end

end;

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

until sy <> comma;

if sy = rparent

then insymbol

else error(4)

end;

if n = 2

then emit(62) {\*PCODE 读入一行\*}

end;

3,4 : begin { write } {\*写入的标准函数\*}

if sy = lparent {\*读入左括号，然后是写入的正式内容\*}

then begin

repeat {\*until读到逗号，一直循环读取字符串\*}

insymbol;

if sy = stringcon {\*写入的是字符串常量，stringcon\*}

then begin

emit1(24,sleng);

emit1(28,inum);

insymbol

end

else begin {\*或者写入的是字符串表达式\*}

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

{\*返回x，内容为处理到“， ： }”之前的内容\*}

if not( x.typ in stantyps ) {\*不再standardtype中，指的是这里的x type不能是array\*}

then error(41);

if sy = colon

then begin

insymbol;

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

if y.typ <> ints

then error(43);

if sy = colon

then begin

if x.typ <> reals

then error(42);

insymbol;

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

if y.typ <> ints

then error(43);

emit(37)

end

else emit1(30,ord(x.typ))

end

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

end

until sy <> comma;

if sy = rparent {\*右括号表示处理完写入的内容\*}

then insymbol

else error(4)

end;

if n = 4

then emit(63)

end; { write }

end { case };

end { standproc } ;

begin { statement }

if sy in statbegsys+[ident]

then case sy of {\*根据symbol不同，采取不同的子过程处理方式\*}

ident : begin {\*如果sy是标识符\*}

i := loc(id);

insymbol;

if i <> 0

then case tab[i].obj of {\*看符号表中该id的object域类型\*}

konstant,typel : error(45);

vvariable: assignment( tab[i].lev,tab[i].adr);

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)

end { case }

end;

beginsy : compoundstatement; {\*得益于之前几百行的努力，这里很方便\*}

ifsy : ifstatement;

casesy : casestatement;

whilesy : whilestatement;

repeatsy: repeatstatement;

forsy : forstatement;

end; { case }

test( fsys, [],14);

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; {\*prt的值是当前符号表的指针\*}

if level > lmax {\*分程序表溢出了，嵌套level过大\*}

then fatal(5);

test([lparent,colon,semicolon],fsys,14); {\*校验合法性\*}

enterblock; {\*记录block信息\*}

prb := b; {\*prb记录的b为当前btab的指针\*}

{\*设置分程序索引表display和tab[prt]的两个域\*}

display[level] := b;

tab[prt].typ := notyp;

tab[prt].ref := prb;

if ( sy = lparent ) and ( level > 1 ) {\*左括号+嵌套level>1，开始处理参数\*}

then parameterlist; {\*具体实现是通过调用parameterlist procedure\*}

btab[prb].lastpar := t; {\*在btab中登记信息\*}

btab[prb].psize := dx;

if isfun {\*如果分程序是function\*}

then if sy = colon {\*读到冒号，接下来应该是返回值类型\*}

then begin

insymbol; { function type }

if sy = ident

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)

end

else skip( [semicolon]+fsys,2 ) {\*读入的不是标识符则会skip\*}

end

else error(5);

if sy = semicolon {\*读到分号说明分程序的声明部分结束\*}

then insymbol

else error(14);

repeat {\*将各类变量登记\*}

if sy = constsy

then constdec;

if sy = typesy

then typedeclaration;

if sy = varsy

then variabledeclaration;

btab[prb].vsize := dx;

while sy in [procsy,funcsy] do

procdeclaration;

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

until sy in statbegsys; {\*直到读到statement语句部分\*}

tab[prt].adr := lc;

insymbol;

statement([semicolon,endsy]+fsys);

while sy in [semicolon]+statbegsys do

begin

if sy = semicolon

then insymbol

else error(14);

statement([semicolon,endsy]+fsys); {\*处理直至分号/出现end的部分\*}

end;

if sy = endsy

then insymbol

else error(57);

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

end { block };

procedure interpret; {\*PCODE的解释执行程序\*}

{\*之前很多看不懂的地方可以参考这里\*}

var ir : order ; { instruction buffer }

pc : integer; { program counter }

t : integer; { top stack index }

b : integer; { base index }

h1,h2,h3: integer;

lncnt,ocnt,blkcnt,chrcnt: integer; { counters }

ps : ( run,fin,caschk,divchk,inxchk,stkchk,linchk,lngchk,redchk );

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

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; {\*dump负责打印程序运行时的现场信息，具体内容见下\*}

var p,h3 : integer;

begin

h3 := tab[h2].lev;

writeln(psout);

writeln(psout);

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

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

writeln(psout,' start of code ',pc:4); {\*打印符号表的域\*}

writeln(psout);

writeln(psout);

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

writeln(psout);

for p := h3 downto 0 do

writeln(psout,p:4,display[p]:6); {\*h3->0，向前打印\*}

writeln(psout);

writeln(psout);

writeln(psout,' top of stack ',t:4,' frame base ':14,b:4); {\*打印栈顶\*}

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; {\*PCODE有很多代码是操作变量值的，相应的分支如下\*}

begin

case ir.f of {\*ir.f即为之前见到的f参数\*}

0 : begin { load addrss }

t := t + 1;

if t > stacksize

then ps := stkchk

else s[t].i := display[ir.x]+ir.y

end;

1 : begin { load value }

t := t + 1;

if t > stacksize

then ps := stkchk

else s[t] := s[display[ir.x]+ir.y]

end;

2 : begin { load indirect }

t := t + 1;

if t > stacksize

then ps := stkchk

else s[t] := s[s[display[ir.x]+ir.y].i]

end;

3 : begin { update display }

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 {\*i域表示整数，r域表示实数\*}

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->char\*}

6 : s[t].i := ord(s[t].c); {\*char->ascii\*}

7 : s[t].c := succ(s[t].c); {\*后继与先前字符\*}

8 : s[t].c := pred(s[t].c);

9 : s[t].i := round(s[t].r);

10 : s[t].i := trunc(s[t].r);

11 : s[t].r := sin(s[t].r);

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 }

end { case ir.y }

end; { inter0 }

procedure inter1; {\*还遇到过许多跳转功能的PCODE\*}

var h3, h4: integer;

begin

case ir.f of

10 : pc := ir.y ; { jump } {\*将pc值设为y的值，相当于跳转\*}

11 : begin { conditional jump }

if not s[t].b {\*带条件的跳转语句\*}

then pc := ir.y;

t := t - 1

end;

12 : begin { switch } {\*没有f=13的case是因为在12中需要查找有f=13指令构成的情况表\*}

h1 := s[t].i;

t := t-1;

h2 := ir.y;

h3 := 0;

repeat

if code[h2].f <> 13

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 } {\*F1U：\*}

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

if h1 <= s[t].i

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

else begin

t := t - 3;

pc := ir.y

end

end;

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

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

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

if h1 <= s[t].i

then begin

s[h2].i := h1;

pc := ir.y

end

else t := t-3;

end;

16 : begin { for1down } {\*F1D，类似上面，增量步长for的循环初始判断\*}

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 }

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

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

if h1 >= s[t].i

then begin

s[h2].i := h1;

pc := ir.y

end

else t := t-3;

end;

18 : begin { mark stack } {\*标记栈\*}

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

if t+h1 > stacksize

then ps := stkchk

else begin

t := t+5;

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

s[t].i := ir.y

end

end;

19 : begin { call }

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

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

h3 := tab[h2].lev;

display[h3+1] := h1;

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

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

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

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

for h3 := t+1 to h4 do

s[h3].i := 0;

b := h1;

t := h4;

pc := tab[h2].adr;

if stackdump

then dump

end;

end { case }

end; { inter1 }

procedure inter2; {\*intpreter2\*}

begin

case ir.f of

20 : begin { index1 }

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

h2 := atab[h1].low;

h3 := s[t].i;

if h3 < h2

then ps := inxchk

else if h3 > atab[h1].high

then ps := inxchk

else begin

t := t-1;

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

end

end;

21 : begin { index }

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 }

h1 := s[t].i;

t := t-1;

h2 := ir.y+t;

if h2 > stacksize

then ps := stkchk

else while t < h2 do

begin

t := t+1;

s[t] := s[h1];

h1 := h1+1

end

end;

23 : begin { copy block }

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 }

t := t+1;

if t > stacksize

then ps := stkchk

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

end;

25 : begin { load real }

t := t+1;

if t > stacksize

then ps := stkchk

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

end;

26 : begin { float }

h1 := t-ir.y;

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

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 }

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 }

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 }

{\*为什么要分成inter123来写case？\*}

procedure inter3;

begin

case ir.f of

30 : begin { write2 }

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;

32 : begin { exit procedure }

t := b-1;

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

b := s[b+3].i

end;

33 : begin { exit function }

t := b;

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

b := s[b+3].i

end;

34 : s[t] := s[s[t].i];

35 : s[t].b := not s[t].b;

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

37 : begin

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

end;

57 : begin

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

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

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)

then ps := redchk

else readln;

63 : begin

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;

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

then begin

writeln(prr);

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

case ps of

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 );

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');

end; { interpret }

procedure setup; {\*设置各类保留字和对应的符号，alng=10\*}

begin

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 ';

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;

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 };

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;

constbegsys := [ plus, minus, intcon, realcon, charcon, ident ];

typebegsys := [ ident, arraysy, recordsy ];

blockbegsys := [ constsy, typesy, varsy, procsy, funcsy, beginsy ];

facbegsys := [ intcon, realcon, charcon, ident, lparent, notsy ];

statbegsys := [ beginsy, ifsy, whilesy, repeatsy, forsy, casesy ];

stantyps := [ notyp, ints, reals, bools, chars ];

lc := 0;

ll := 0;

cc := 0;

ch := ' ';

errpos := 0;

errs := [];

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

writeln;

write( 'Source input file ?');

readln( inf );

assign( psin, inf );

reset( psin );

write( 'Source listing file ?');

readln( outf );

assign( psout, outf );

rewrite( psout );

assign ( prd, 'con' );

write( 'result file : ' );

readln( fprr );

assign( prr, fprr );

reset ( prd );

rewrite( prr );

t := -1;

a := 0;

b := 1;

sx := 0;

c2 := 0;

display[0] := 1;

iflag := false;

oflag := false;

skipflag := false;

prtables := false;

stackdump := false;

insymbol;

if sy <> programsy

then error(3)

else begin

insymbol;

if sy <> ident

then error(2)

else begin

progname := id;

insymbol;

if sy <> lparent

then error(9)

else repeat

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

then insymbol

else error(4);

if not oflag then error(20)

end

end;

enterids;

with btab[1] do

begin

last := t;

lastpar := 1;

psize := 0;

vsize := 0;

end;

block( blockbegsys + statbegsys, false, 1 );

if sy <> period

then error(2);

emit(31); { halt }

if prtables

then printtables;

if errs = []

then interpret

else begin

writeln( psout );

writeln( psout, 'compiled with errors' );

writeln( psout );

errormsg;

end;

writeln( psout );

close( psout );

close( prr )

end.