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 ot 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 } {\*key word应当理解为保留字\*}

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. } {\*最大错误数量\*}

omax = 63; { highest order code }

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

nmax = 32767; { 2\*\*15-1 } {\*数字的范围\*}

lineleng = 132; { output line length }

linelimit = 200; {\*行数限制\*}

stacksize = 1450; {\*数据栈大小\*}

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);

index = -xmax..+xmax;

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

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

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

symset = set of symbol;

typset = set of types;

item = record

typ: types;

ref: index;

end;

order = packed record

f: -omax..+omax;

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 }

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

sy: symbol; { last symbol read by insymbol }

errs: set of 0..ermax; {\*记录错误的集合\*}

id: alfa; { identifier from insymbol }

progname: alfa;

stantyps: typset;

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

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

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

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

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

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

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

packed record

name: alfa;

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 } {\*分符号表\*}

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; {\*P代码表\*}

psin,psout,prr,prd:text; { default in pascal p } {\*写入inf,outf,fppr文件的文本\*}

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

procedure errormsg; {\*打印错误信息摘要的过程\*}

var k : integer;

msg: array[0..ermax] of alfa; {\*给定错误信息表,最多ermax种错误\*}

begin

msg[0] := 'undef id '; msg[1] := 'multi def '; {\*给定错误类型'k',及其提示信息\*}

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); {\*向文件中打印一个空行\*}

writeln(psout,'key words'); {\*向psout文件中输出'key words',并换行\*}

k := 0;

while errs <> [] do {\*如果还有错误信息没有处理\*}

begin

while not( k in errs )do k := k + 1; {\*如果不存在第k种错误,则判断是否存在地k+1中\*}

writeln(psout, k, ' ', msg[k] ); {\*在文件中输出错误的编号及其信息\*}

errs := errs - [k] {\*将错误集合中的该类错误去除(因为已经处理过)\*}

end { while errs } {\*循环直到所有错误被处理\*}

end { errormsg } ;

procedure endskip; {\*源程序出错后再整个跳过部分代码下面画下划线\*}

begin { underline skipped part of input }

while errpos < cc do

begin

write( psout, '-');

errpos := errpos + 1

end;

skipflag := false

end { endskip };

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

begin

if cc = ll {\*如果读到了一行的末尾\*}

then begin

if eof( psin ) {\*文件读完了\*}

then begin

writeln( psout ); {\*写输出文件\*}

writeln( psout, 'program incomplete' ); {\*提示信息\*}

errormsg; {\*输出错误提示信息到list文件\*}

exit;

end;

if errpos <> 0 {\*说明有错误,开始错误处理\*}

then begin

if skipflag then endskip; {\*跳过错误代码\*}

writeln( psout );

errpos := 0

end;

write( psout, lc: 5, ' '); {\*没有错误执行的操作,在list文件中输出当前PCODE的行数以及一个空格,不换行\*}

ll := 0; {\*将行长度和行指针置零\*}

cc := 0;

while not eoln( psin ) do {\*如果文件没有读完,读下一行\*}

begin

ll := ll + 1; {\*统计行的长度\*}

read( psin, ch ); {\*读取下一个字符\*}

write( psout, ch ); {\*输出到list文件中\*}

line[ll] := ch {\*将ch保存到line中,循环结束line保存下一行代码的所有信息\*}

end;

ll := ll + 1;

readln( psin );

line[ll] := ' '; {\*一行的末尾置为空格\*}

writeln( psout );

end;

cc := cc + 1; {\*行指针前移\*}

ch := line[cc]; {\*取词\*}

end { nextch };

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

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 ); {\*打印表格溢出信息,写入数据多于表大小时会终止程序\*}

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

begin

writeln( psout );

errormsg;

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

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

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

msg[7] := 'strings ';

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

exit; {terminate compilation }

end { fatal };

procedure insymbol; {reads next symbol} {\*取符号方法\*}

label 1,2,3; {\*定义label,为goto的使用做准备\*}

var i,j,k,e: integer;

procedure readscale; {\*处理实数的指数部分\*}

var s,sign: integer;

begin

nextch;

sign := 1; {\*符号\*}

s := 0; {\*数字\*}

if ch = '+' {\*如果读到'+',不作处理\*}

then nextch

else if ch = '-' {\*如果是'-',符号设为负\*}

then begin

nextch;

sign := -1

end;

if not(( ch >= '0' )and (ch <= '9' )) {\*如果符号后面跟的不是数字,报错\*}

then error( 40 )

else repeat

s := 10\*s + ord( ord(ch)-ord('0')); {\*把数字存到s中\*}

nextch;

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

e := s\*sign + e {\*和下面计算中的e结合得到真的e\*}

end { readscale };

procedure adjustscale; {\*根据小数位数和指数大小求出数字数值的大小\*}

var s : integer;

d, t : real;

begin

if k + e > emax {\*当前的位数加上指数如果超上限报错\*}

then error(21)

else if k + e < emin {\*小于最小值\*}

then rnum := 0 {\*精度不够了,直接记为零\*}

else begin

s := abs(e);

t := 1.0;

d := 10.0;

repeat

while not odd(s) do {\*把偶次幂先用平方处理完\*}

begin

s := s div 2;

d := sqr(d) {\*sqr表示平方\*}

end;

s := s - 1;

t := d \* t {\*在乘一下自己,完成1次,即将e分解为2N+1或2N的形式\*}

until s = 0; {\*t此时为10的e次方\*}

if e >= 0

then rnum := rnum \* t {\*e大于零就乘10的e次方\*}

else rnum := rnum / t {\*反之除\*}

end

end { adjustscale };

procedure options; {\*编译选项\*}

procedure switch( var b: boolean ); {\*处理编译选项中的'+''-'号\*}

begin

b := ch = '+'; {\*判断当前符号是否为'+'并存入b中返回,注意pascal中变量形参传的是地址\*}

if not b {\*如果不是加号\*}

then if not( ch = '-' ) {\*如果也不是减号\*}

then begin { print error message } {\*输出错误信息\*}

while( ch <> '\*' ) and ( ch <> ',' ) do {\*跳过无用符号\*}

nextch;

end

else nextch

else nextch

end { switch };

begin { options } {\*处理编译选项\*}

repeat

nextch;

if ch <> '\*' {\*编译选项为\*$t+,s+\*的形式\*}

then begin

if ch = 't' {\*字母t表示与打印相关的操作\*}

then begin

nextch;

switch( prtables ) {\*根据符号判断是否打印表格\*}

end

else if ch = 's' {\*s表示卸出打印\*}

then begin

nextch;

switch( stackdump )

end;

end

until ch <> ','

end { options };

begin { insymbol }

1: while( ch = ' ' ) or ( ch = chr(9) ) do {\*第一个flag立起来了! chr可以获得9号字符,即跳过所有的空格和\t\*}

nextch; { space & htab }

case ch of

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

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

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

begin { identifier of wordsymbol } {\*如果是字母,开始识别单词\*}

k := 0;

id := ' ';

repeat

if k < alng {\*alng是限定的关键词长度\*}

then begin

k := k + 1;

id[k] := ch

end;

nextch

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

i := 1;

j := nkw; { binary search } {\*二分查表,找到当前id在表中的位置\*}

repeat

k := ( i + j ) div 2;

if id <= key[k]

then j := k - 1;

if id >= key[k]

then i := k + 1;

until i > j;

if i - 1 > j

then sy := ksy[k] {\*获取当前ID对应的sym\*}

else sy := ident {\*没有找到即为标识符\*}

end;

'0','1','2','3','4','5','6','7','8','9': {\*数字开始当做数字识别\*}

begin { number }

k := 0;

inum := 0;

sy := intcon; {\*sy设为intcon表示数字\*}

repeat

inum := inum \* 10 + ord(ch) - ord('0'); {\*把整数部分读完,存到inum\*}

k := k + 1; {\*k统计当前数字位数\*}

nextch

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

if( k > kmax ) or ( inum > nmax ) {\*超上限报错\*}

then begin

error(21);

inum := 0;

k := 0

end;

if ch = '.' {\*开始读小数\*}

then begin

nextch;

if ch = '.'

then ch := ':'

else begin

sy := realcon; {\*sym为实数\*}

rnum := inum; {\*rnum存实数的值\*}

e := 0; {\*指数\*}

while ( ch >= '0' ) and ( ch <= '9' ) do {\*把数字读完\*}

begin

e := e - 1;

rnum := 10.0 \* rnum + (ord(ch) - ord('0')); {\*暂时当做整数存\*}

nextch

end;

if e = 0 {\*小数点后没数字,40号error\*}

then error(40);

if ch = 'e' {\*如果是科学计数法\*}

then readscale; {\*算e\*}

if e <> 0 then adjustscale {\*算数,rnum存数\*}

end

end

else if ch = 'e'

then begin

sy := realcon;

rnum := inum;

e := 0;

readscale;

if e <> 0

then adjustscale

end;

end;

':':

begin

nextch;

if ch = '='

then begin

sy := becomes;

nextch

end

else sy := colon

end;

'<':

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;

'.':

begin

nextch;

if ch = '.'

then begin

sy := colon; {\*..居然算作colon冒号\*}

nextch

end

else sy := period

end;

'''': {\*当前字符是否单引号\*}

begin

k := 0;

2: nextch;

if ch = ''''

then begin

nextch;

if ch <> ''''

then goto 3

end;

if sx + k = smax

then fatal(7);

stab[sx+k] := ch;

k := k + 1;

if cc = 1

then begin { end of line }

k := 0;

end

else goto 2;

3: if k = 1 {\*双引号中间只有一个字符\*}

then begin

sy := charcon; {\*sym类型为字符类型\*}

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

end

else if k = 0 {\*空引号,中间没东西\*}

then begin

error(38); {\*报错\*}

sy := charcon; {\*类型字符常量\*}

inum := 0 {\*asc为0\*}

end

else begin

sy := stringcon; {\*否则就是一个字符串类型\*}

inum := sx;

sleng := k;

sx := sx + k

end

end;

'(':

begin

nextch;

if ch <> '\*'

then sy := lparent

else begin { comment }

nextch;

if ch = '$'

then options;

repeat

while ch <> '\*' do nextch;

nextch

until ch = ')';

nextch;

goto 1

end

end;

'{':

begin

nextch;

if ch = '$' {\*左括号加$是进行编译选项的设置\*}

then options;

while ch <> '}' do

nextch;

nextch;

goto 1

end;

'+', '-', '\*', '/', ')', '=', ',', '[', ']', ';': {\*操作符直接处理\*}

begin

sy := sps[ch];

nextch

end;

'$','"' ,'@', '?', '&', '^', '!': {\*单独出现算错\*}

begin

error(24);

nextch;

goto 1

end

end { case }

end { insymbol };

procedure enter(x0:alfa; x1:objecttyp; x2:types; x3:integer ); {\*将当前符号(分程序外的)录入符号表\*}

begin

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

with tab[t] do

begin

name := x0;

link := t - 1;

obj := x1;

typ := x2;

ref := 0;

normal := true;

lev := 0;

adr := x3;

end

end; { enter }

procedure enterarray( tp: types; l,h: integer ); {\*将数组信息录入数组表atab\*}

begin

if l > h {\*下界大于上界,错误\*}

then error(27);

if( abs(l) > xmax ) or ( abs(h) > xmax )

then begin

error(27);

l := 0;

h := 0;

end;

if a = amax {\*表满了\*}

then fatal(4)

else begin

a := a + 1;

with atab[a] do

begin

inxtyp := tp; {\*下标类型\*}

low := l; {\*上界和下界\*}

high := h

end

end

end { enterarray };

procedure enterblock; {\*将分程序登录到分程序表\*}

begin

if b = bmax {\*表满了\*}

then fatal(2) {\*报错退出\*}

else begin

b := b + 1;

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

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

end

end { enterblock };

procedure enterreal( x: real ); {\*登陆实常量表\*}

begin

if c2 = c2max - 1

then fatal(3)

else begin

rconst[c2+1] := x;

c1 := 1;

while rconst[c1] <> x do

c1 := c1 + 1;

if c1 > c2

then c2 := c1

end

end { enterreal };

procedure emit( fct: integer ); {\*emit和下面两个方法都是用来生成PCODE的,后面接的数字是代表有几个操作数\*}

begin

if lc = cmax

then fatal(6);

code[lc].f := fct;

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; {\*打印表的过程\*}

var i: integer;

o: order;

mne: array[0..omax] of

packed array[1..5] of char;

begin

mne[0] := 'LDA '; mne[1] := 'LOD '; mne[2] := 'LDI '; {\*定义PCODE指令符\*}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

writeln(psout);

writeln(psout);

writeln(psout);

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

writeln(psout);

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

with tab[i] do

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

writeln( psout );

writeln( psout );

writeln( psout );

writeln( psout, 'blocks last lpar psze vsze' );

writeln( psout );

for i := 1 to b do

with btab[i] do

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

writeln( psout );

writeln( psout );

writeln( psout );

writeln( psout, 'arrays xtyp etyp eref low high elsz size');

writeln( psout );

for i := 1 to a do

with atab[i] do

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

writeln( psout );

writeln( psout );

writeln( psout );

writeln( psout, 'code:');

writeln( psout );

for i := 0 to lc-1 do

begin

write( psout, i:5 );

o := code[i];

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

if o.f < 31

then if o.f < 4

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

else write( psout, o.y:10 )

else write( psout, ' ' );

writeln( psout, ',' )

end;

writeln( psout );

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

end { printtables };

procedure block( fsys: symset; isfun: boolean; level: integer ); {\*程序分析过程\*}

type conrec = record {\*这种结构体可以根据不同的type类型来保存不同样式的数据\*}

case tp: types of

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

reals :( r:real )

end;

var dx : integer ; { data allocation index }

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

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

x : integer ;

procedure skip( fsys:symset; n:integer); {\*跳过错误的代码段\*}

begin

error(n);

skipflag := true;

while not ( sy in fsys ) do

insymbol;

if skipflag then endskip

end { skip };

procedure test( s1,s2: symset; n:integer ); {\*检查当前sym是否合法\*}

begin

if not( sy in s1 )

then skip( s1 + s2, n )

end { test };

procedure testsemicolon; {\*检查分号是否合法\*}

begin

if sy = semicolon

then insymbol

else begin

error(14);

if sy in [comma, colon]

then insymbol

end;

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

end { testsemicolon };

procedure enter( id: alfa; k:objecttyp ); {\*将分程序中的某一符号入符号表\*}

var j,l : integer;

begin

if t = tmax {\*表满了报错退出\*}

then fatal(1)

else begin

tab[0].name := id;

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

l := j;

while tab[j].name <> id do

j := tab[j].link;

if j <> 0 {\*j不等于0说明此符号已经在符号表中出现过,报1号错误,意味着重复定义了\*}

then error(1)

else begin {\*没重复定义就正常入栈\*}

t := t + 1;

with tab[t] do {\*将符号放入符号表,注意这里并没有给定符号的typ,ref和adr,这三个变量在procedure typ中被处理\*}

begin

name := id; {\*输入参数之一,符号的名字\*}

link := l;

obj := k; {\*输入参数之一,符号代表的目标种类(大类)\*}

typ := notyp;

ref := 0;

lev := level;

adr := 0;

normal := false { initial value }

end;

btab[display[level]].last := t {\*更新当前层最后一个标识符\*}

end

end

end { enter };

function loc( id: alfa ):integer; {\*查找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 {\*符号没找到,说明之前没声明,报0号错误\*}

then error(0);

loc := j

end { loc } ;

procedure entervariable; {\*变量登陆符号表的过程\*}

begin

if sy = ident

then begin

enter( id, vvariable );

insymbol

end

else error(2)

end { entervariable };

procedure constant( fsys: symset; var c: conrec ); {\*处理程序中出现的常量,变量c负责返回该常量的类型和值\*}

var x, sign : integer;

begin

c.tp := notyp;

c.i := 0;

test( constbegsys, fsys, 50 );

if sy in constbegsys {\*如果第一个sym是常量开始的符号,才往下继续分析\*}

then begin {\*根据不同的符号执行不同的操作,目的就是返回正确的c\*}

if sy = charcon {\*对字符常量\*}

then begin

c.tp := chars; {\*类型是char\*}

c.i := inum; {\*inum存储该字符的ascii码值\*}

insymbol {\*获取下一个sym\*}

end

else begin

sign := 1; {\*不是符号常量\*}

if sy in [plus, minus]

then begin

if sy = minus

then sign := -1; {\*负号变符号\*}

insymbol

end;

if sy = ident {\*遇到了标识符\*}

then begin

x := loc(id); {\*找到当前id在表中的位置\*}

if x <> 0 {\*找到了\*}

then

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

then error(25)

else begin

c.tp := tab[x].typ; {\*获得常量类型\*}

if c.tp = reals {\*对实数和整数采取不同的赋值方法\*}

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

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

end;

insymbol

end

else if sy = intcon {\*遇到整数\*}

then begin

c.tp := ints; {\*存type存值\*}

c.i := sign\*inum;

insymbol

end

else if sy = realcon {\*遇到实数\*}

then begin

c.tp := reals;

c.r := sign\*rnum;

insymbol

end

else skip(fsys,50) {\*跳过无用符号\*}

end;

test(fsys,[],6)

end

end { constant };

procedure typ( fsys: symset; var tp: types; var rf,sz:integer ); {\*处理类型说明,返回当前关键词的类型,在符号表中的位置,以及需要占用存储空间的大小\*}

var eltp : types; {\*元素类型\*}

elrf, x : integer;

elsz, offset, t0, t1 : integer;

procedure arraytyp( var aref, arsz: integer ); {\*处理数组类型的子过程\*}

var eltp : types; {\*记录元素的类型,pascal中一个数组的所有元素的类型必须相同\*}

low, high : conrec; {\*记录数组编号(index)的上下界\*}

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

begin

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

if low.tp = reals {\*如果下界类型为实型\*}

then begin

error(27); {\*报27号错误\*}

low.tp := ints; {\*类型为整型\*}

low.i := 0 {\*数值设为0\*}

end;

if sy = colon {\*下界后面跟'..',类型是colon,constant结束后读入了下一个sym\*}

then insymbol {\*获得下一个sym\*}

else error(13); {\*如果后面跟的不是..,报13号错误\*}

constant( [rbrack, comma, rparent, ofsy ] + fsys, high ); {\*获取数组下表上界\*}

if high.tp <> low.tp {\*上下界类型不同报错,也就是说上界也必须是整型\*}

then begin

error(27); {\*报27号错误\*}

high.i := low.i {\*容错,是使得上界等于下界\*}

end;

enterarray( low.tp, low.i, high.i ); {\*将数组的信息录入到atab中\*}

aref := a; {\*获取当前数组在atab中的位置\*}

if sy = comma {\*后面接逗号,说明需要建立多维数组\*}

then begin

insymbol; {\*读取下一个字符\*}

eltp := arrays; {\*数组中的每个元素类型都是数组\*}

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

end

else begin

if sy = rbrack {\*遇到右中括号,则index部分声明完毕\*}

then insymbol {\*获取下一个sym\*}

else begin

error(12); {\*缺少右中括号\*}

if sy = rparent {\*如果是右括号\*}

then insymbol {\*容错\*}

end;

if sy = ofsy {\*获取到了of关键字\*}

then insymbol {\*获取下一个sym\*}

else error(8); {\*没有of报8号错\*}

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

end;

with atab[aref] do {\*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中登陆的位置\*}

elsize := elsz {\*记录每个元素的大小\*}

end

end { arraytyp };

begin { typ } {\*类型处理过程开始\*}

tp := notyp; {\*用以存储变量的类型\*}

rf := 0; {\*用以记录符号在符号表中的位置\*}

sz := 0; {\*用以储存该类型的大小\*}

test( typebegsys, fsys, 10 ); {\*测试当前符号是否是数组声明的开始符号,如果不是则报10号错误\*}

if sy in typebegsys {\*如果是数组声明的开始符号\*}

then begin

if sy = ident {\*如果现在的符号是标识符\*}

then begin

x := loc(id); {\*查找id在符号表中的位置\*}

if x <> 0 {\*如果找到了\*}

then with tab[x] do {\*对其对应表项进行操作\*}

if obj <> typel {\*标识符的种类不是'种类'(typel)\*}

then error(29) {\*报29号错,因为声明一个变量需要先标明其类型\*}

else begin

tp := typ; {\*获得其代表的类型(char,int,real..)\*}

rf := ref; {\*获得其在符号表中的位置\*}

sz := adr; {\*获得其在运行栈中分配的储存单元的相对地址\*}

if tp = notyp {\*如果未定义类型\*}

then error(30) {\*报30号错\*}

end;

insymbol {\*获得下一个sym\*}

end

else if sy = arraysy {\*如果遇到的是数组元素,即声明开头为'array'\*}

then begin

insymbol; {\*获得下一个sym\*}

if sy = lbrack {\*数组元素声明应该从左中括号开始,即表明数组的大小/维度\*}

then insymbol {\*获取下一个sym\*}

else begin {\*如果不是左中括号开始\*}

error(11); {\*报11号错误,说明左括号发生错误\*}

if sy = lparent {\*如果找到了左括号,可能是用户输入错误,报错后做容错处理\*}

then insymbol {\*获取下一个sym\*}

end;

tp := arrays; {\*当前类型设置为数组类型\*}

arraytyp(rf,sz) {\*获得数组在atab表中的登陆位置,和数组的大小\*}

end

else begin { records } {\*否则一定是record的类型,因为typebegsys中只包含ident,arraysy和recordsy三种类型\*}

insymbol; {\*获取下一个sym\*}

enterblock; {\*登陆子程序\*}

tp := records; {\*当前类型设置为records类型\*}

rf := b; {\*rf指向当前过程在block表中的位置\*}

if level = lmax {\*如果当前嵌套层次已经是最大层次了,即不能产生更深的嵌套\*}

then fatal(5); {\*报5号严重错误并终止程序\*}

level := level + 1; {\*如果还能嵌套,声明程序成功,block的层次是当前层次+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 {\*end之前都是记录类型变量内的变量声明\*}

begin { field section } {\*开始处理record内部的成员变量\*}

if sy = ident {\*如果遇到的是标识符\*}

then begin

t0 := t; {\*获得当前tab指针的位置\*}

entervariable; {\*变量入表\*}

while sy = comma do {\*同种变量之间通过逗号分隔,未遇到分号则继续读入\*}

begin

insymbol; {\*获得下一个sym\*}

entervariable {\*继续变量入表的过程\*}

end;

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

then insymbol {\*获取sym\*}

else error(5); {\*如果没有遇到逗号或者冒号,则抛出5号错误\*}

t1 := t; {\*记录当前tab栈顶符号的位置,至此t0到t1的符号表中并没有填写typ,ref和adr\*}

typ( fsys + [semicolon, endsy, comma,ident], eltp, elrf,elsz ); {\*递归调用typ来处理记录类型的成员变量,确定各成员的类型,ref和adr(注意对于不同的类型,ref和adr可能表示不同的意义)\*}

while t0 < t1 do {\*填写t0到t1中信息缺失的部分,需要注意的是t0~t1都是同一类型的变量,因此size大小是相同的\*}

begin

t0 := t0 + 1; {\*指针上移\*}

with tab[t0] do {\*修改当前表项\*}

begin

typ := eltp; {\*给typ赋值,eltp来之上面递归调用的typ语句\*}

ref := elrf; {\*给ref赋值\*}

normal := true; {\*给normal标记赋值,所有normal的初值都是false\*}

adr := offset; {\*记录该变量相对于起始地址的位移\*}

offset := offset + elsz {\*获得下一变量的其实地址\*}

end

end

end; { sy = ident }

if sy <> endsy {\*遇到end说明成员声明已经结束了\*}

then begin

if sy = semicolon {\*end后面需要接分号\*}

then insymbol {\*获取下一个sym\*}

else begin {\*如果接的不是分号\*}

error(14); {\*先报个错\*}

if sy = comma {\*如果是逗号做容错处理\*}

then insymbol {\*然后获取下一个sym类型\*}

end;

test( [ident,endsy, semicolon],fsys,6 ) {\*检验当前符号是否合法\*}

end

end; { field section }

btab[rf].vsize := offset; {\*offset存储了当前的局部变量,参数以及display区所占的空间总数,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 ) {\*检查当前sym是否合法\*}

end;

end { typ };

procedure parameterlist; { formal parameter list } {\*处理过程或函数说明中的形参,将形参登陆到符号表\*}

var tp : types; {\*记录类型\*}

valpar : boolean; {\*记录当前参数是否为值形参(valueparameter)\*}

rf, sz, x, t0 : integer;

begin

insymbol; {\*获得下一个sym\*}

tp := notyp; {\*初始化类型\*}

rf := 0; {\*初始化符号表位置\*}

sz := 0; {\*初始化元素大小\*}

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

while sy in [ident, varsy] do {\*如果当前的符号是标识符或者var关键字\*}

begin

if sy <> varsy {\*如果是var关键字\*}

then valpar := true {\*valpar标识符设置为真\*}

else begin

insymbol; {\*如果不是标识符,获取下一个sym\*}

valpar := false {\*将valpar设置为假\*}

end;

t0 := t; {\*记录当前符号表栈顶位置\*

entervariable; {\*调用变量入表的子过程,将参数符号放入符号表\*}

while sy = comma do {\*如果识别到逗号,说明还有同类型的参数,继续放入符号表\*}

begin

insymbol; {\*获取下一个sym\*}

entervariable; {\*将当前sym放入符号表\*}

end;

if sy = colon {\*如果识别到冒号,开始处理类型\*}

then begin

insymbol; {\*获取下一个sym,这里应当是类型\*}

if sy <> ident {\*如果不是标识符\*}

then error(2) {\*报2号错误\*}

else begin

x := loc(id); {\*如果是标识符,则寻找其在符号表中的位置\*}

insymbol; {\*获取下一个sym\*}

if x <> 0 {\*如果在符号表中找到了sym\*}

then with tab[x] do {\*对当前表项做操作\*}

if obj <> typel {\*如果当前的符号不是类型标识符\*}

then error(29) {\*报29号错误\*}

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 ) {\*检验当前符号是否合法,不合法报14号错误\*}

end

else error(5); {\*如果不是分号,报5号错误\*}

while t0 < t do {\*t0~t都是同一类型将上面处理的符号中的属性填写完整\*}

begin

t0 := t0 + 1; {\*获得刚才读到的第一个参数\*}

with tab[t0] do {\*对当前符号表中的符号做操作\*}

begin

typ := tp; {\*设置当前符号的类型\*}

ref := rf; {\*设置当前符号在符号表中的位置\*}

adr := dx; {\*设置形参的相对地址\*}

lev := level; {\*设置形参的level\*}

normal := valpar; {\*设置当前变量的normal标记\*}

dx := dx + sz {\*更新位移量\*}

end

end;

if sy <> rparent {\*如果声明结束之后不是右括号\*}

then begin

if sy = semicolon {\*而是分号,说明还有需要声明的参数\*}

then insymbol {\*获取下一个sym\*}

else begin

error(14); {\*否则报14号错误\*}

if sy = comma {\*如果是逗号,做容错处理\*}

then insymbol {\*接受下一个sym\*}

end;

test( [ident, varsy],[rparent]+fsys,6) {\*检查下面的符号是否是标识符或者变量声明,均不是则报6号错误\*}

end

end { while };

if sy = rparent {\*参数声明结束后应当用右括号结尾\*}

then begin

insymbol; {\*获取下一个符号\*}

test( [semicolon, colon],fsys,6 ) {\*声明结束后用分号结束或使用冒号声明返回值类型,如果不是这两种符号,报6号错误\*}

end

else error(4) {\*不是右括号结尾,报错\*}

end { parameterlist };

procedure constdec; {\*常量声明的处理过程\*}

var c : conrec;

begin

insymbol; {\*获取下一个sym\*}

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

while sy = ident do {\*当获得的是标志符的是否做循环\*}

begin

enter(id, konstant); {\*入表,类型为konstant表示常量\*}

insymbol;

if sy = eql {\*等号\*}

then insymbol

else begin

error(16);

if sy = becomes {\*赋值符号容错\*}

then insymbol

end;

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

tab[t].typ := c.tp; {\*填表\*}

tab[t].ref := 0; {\*常量ref为0\*}

if c.tp = reals

then begin {\*实型和整型的操作不同\*}

enterreal(c.r);

tab[t].adr := c1; {\*实常量的adr保存了其在rconst表中的登陆的位置\*}

end

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

testsemicolon

end

end { constdec };

procedure typedeclaration; {\*处理类型声明\*}

var tp: types;

rf, sz, t1 : integer;

begin

insymbol;

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

while sy = ident do {\*对于是标识符的情况进行操作\*}

begin

enter(id, typel); {\*类型的名称的类型入表\*}

t1 := t; {\*获得符号表顶部指针\*}

insymbol;

if sy = eql {\*获取等号\*}

then insymbol

else begin

error(16);

if sy = becomes {\*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 {\*将返回值填表\*}

begin

typ := tp;

ref := rf;

adr := sz

end;

testsemicolon

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

begin

t0 := t;

entervariable;

while sy = comma do

begin

insymbol;

entervariable; {\*调用变量入表的程序\*}

end;

if sy = colon

then insymbol

else error(5);

t1 := t;

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

while t0 < t1 do

begin

t0 := t0 + 1;

with tab[t0] do {\*填表\*}

begin

typ := tp;

ref := rf;

lev := level;

adr := dx;

normal := true;

dx := dx + sz

end

end;

testsemicolon

end

end { variabledeclaration };

procedure procdeclaration; {\*处理过程声明\*}

var isfun : boolean;

begin

isfun := sy = funcsy;

insymbol;

if sy <> ident

then begin

error(2);

id :=' '

end;

if isfun {\*函数和过程使用不同的kind类型\*}

then enter(id,funktion)

else enter(id,prozedure);

tab[t].normal := true;

insymbol;

block([semicolon]+fsys, isfun, level+1 ); {\*过程的处理直接调用block\*}

if sy = semicolon

then insymbol

else error(14);

emit(32+ord(isfun)) {exit} {\*推出过程/函数\*}

end { proceduredeclaration };

procedure statement( fsys:symset );

var i : integer;

procedure expression(fsys:symset; var x:item); forward; {\*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,做分号或句号之一\*}

repeat

if sy = period {\*如果当前的符号是句号,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 } {\*处理成员变量\*}

if sy <> ident {\*如果获取到的不是标识符\*}

then error(2) {\*报2号错误\*}

else begin

if v.typ <> records {\*如果处理的不是记录类型\*}

then error(31) {\*报31号错误\*}

else begin { search field identifier } {\*在符号表中寻找类型标识符\*}

j := btab[v.ref].last; {\*获得该结构体在符号表中最后一个符号的位置\*}

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 {\*在当前层(记录中)没找到对应的符号,符号未声明\*}

then error(0); {\*报0号错误\*}

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) {\*生成一条指令来计算此位移\*}

end;

insymbol {\*获取下一个sym\*}

end

end

else begin { array selector } {\*处理数组下表\*}

if sy <> lbrack {\*如果下表不是左括号开头\*}

then error(11); {\*报11号错误\*}

repeat {\*循环,针对多维数组\*}

insymbol; {\*获取下一个sym\*}

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

if v.typ <> arrays {\*如果传入的类型不是数组\*}

then error(28) {\*报22号错误\*}

else begin

a := v.ref; {\*获得该数组在atab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\*}

then error(26) {\*报26号错误\*}

else if atab[a].elsize = 1 {\*如果是变量形参\*}

then emit1(20,a) {\*进行寻址操作\*}

else emit1(21,a); {\*对值形参也进行寻址操作\*}

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,说明没有更高维的数组\*}

if sy = rbrack {\*如果读到右中括号\*}

then insymbol {\*读取下一个sym\*}

else begin

error(12); {\*没读到右中括号则报12号错误\*}

if sy = rparent {\*如果读到了右括号,做容错处理\*}

then insymbol {\*读取下一个sym\*}

end

end

until not( sy in[lbrack, lparent, period]); {\*循环直到所有子结构(数组下标或者记录)都被识别完位置\*}

test( fsys,[],6) {\*检测当前的符号是否合法\*}

end { selector };

procedure call( fsys: symset; i:integer ); {\*处理非标准过程和函数调用的方法,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 } {\*生成标记栈指令,传入被调用过程或函数在tab表中的位置,建立新的内务信息区\*}

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

cp := i; {\*记录被调用过程在符号表中的位置\*}

if sy = lparent {\*如果是识别到左括号\*}

then begin { actual parameter list } {\*开始处理参数\*}

repeat {\*开始循环\*}

insymbol; {\*获取参数的sym\*}

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

then error(39) {\*报39号错误\*}

else begin {\*开始处理参数\*}

cp := cp + 1; {\*将cp指针向上移动一格\*}

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 } {\*开始处理值形参\*}

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相同\*}

then begin

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

then error(36) {\*报36号错误\*}

else if x.typ = arrays {\*如果遇到了数组类型\*}

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

else if x.typ = records {\*如果遇到了记录类型\*}

then emit1(22,btab[x.ref].vsize) {\*同样生成装入块指令完成操作,只是细节有所不同\*}

end

else if ( x.typ = ints ) and ( tab[cp].typ = reals ) {\*如果表达式的类型是整型,但是要求是输入的是实型参数\*}

then emit1(26,0) {\*生成26号指令,进行类型转换\*}

else if x.typ <> notyp {\*如果没有获取到表达式的类型\*}

then error(36); {\*报36号错,参数类型异常\*}

end

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

if sy <> ident {\*变量形参应该先识别到标识符\*}

then error(2) {\*若不是标识符开头,报2号错\*}

else begin {\*如果是标识符开头\*}

k := loc(id); {\*找到当前id在表中的位置\*}

insymbol; {\*获取下一个符号\*}

if k <> 0 {\*在符号表中找到了id\*}

then begin

if tab[k].obj <> vvariable {\*如果获取到的形参类型不是变量类型\*}

then error(37); {\*报37号错\*}

x.typ := tab[k].typ; {\*否则记录当前的符号类型\*}

x.ref := tab[k].ref; {\*记录当前参数指向的btab的位置\*}

if tab[k].normal {\*如果是值形参\*}

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

else emit2(1,tab[k].lev,tab[k].adr); {\*将变量的值装入栈顶(对应变量形参)\*}

if sy in [lbrack, lparent, period] {\*如果后面跟的可以是做中括号(数组下标),左括号(容错)或句号(对应记录)\*}

then

selector(fsys+[comma,colon,rparent],x); {\*调用分析子结构的过程来处理\*}

if ( x.typ <> tab[cp].typ ) or ( x.ref <> tab[cp].ref ) {\*如果参数的符号类型或所在表中的位置和符号表中记录的不同\*}

then error(36) {\*报36号错误\*}

end

end

end {variable parameter }

end;

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

until sy <> comma; {\*直到出现的不是都好,说明参数声明结束了\*}

if sy = rparent {\*补齐右括号\*}

then insymbol {\*获取下一个sym\*}

else error(4) {\*没有右括号,报4号错误\*}

end;

if cp < lastp {\*如果当前符号的位置没有到达最后一个符号的位置\*}

then error(39); { too few actual parameters } {\*报39号错误,说明符号没有处理完\*}

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

if tab[i].lev < level {\*如果符号所在层次小于当前层次\*}

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

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33号错误\*}

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 {\*第一个是int\*}

then if b = ints {\*第二个也是int\*}

then resulttype := ints {\*返回int类型\*}

else begin

resulttype := reals; {\*否则结果为real\*}

emit1(26,1) {\*并对a进行类型转化\*}

end

else begin

resulttype := reals; {\*第一个是real,则返回real\*}

if b = ints {\*如果第二个是int\*}

then emit1(26,0) {\*对b进行转化\*}

end

end { resulttype } ;

procedure expression( fsys: symset; var x: item ); {\*处理表达式的过程,返回类型和在表中的位置\*}

var y : item;

op : symbol;

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

var y : item;

op : symbol;

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

var y : item;

op : symbol;

procedure factor( fsys: symset; var x: item );{\*处理因子的子过程\*}

var i,f : integer;

procedure standfct( n: integer ); {\*处理标准函数的子过程，传入标准函数的编号n，执行不同的操作\*}

var ts : typset; {\*类型集合\*}

begin { standard function no. n }

if sy = lparent {\*如果当前的符号是左括号\*}

then insymbol {\*获取下一个sym\*}

else error(9); {\*如果当前符号不是左括号,报9号错误提示左括号出错\*}

if n < 17 {\*如果标准函数的编号小于17\*}

then begin

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

case n of {\*根据不同的函数编号来进行操作\*}

{ abs, sqr } 0,2: begin {\*如果是0,2号操作,完成求绝对值和平方\*}

ts := [ints, reals]; {\*定义符号集合为整型和实型\*}

tab[i].typ := x.typ; {\*函数的返回值类型\*}

if x.typ = reals {\*如果参数类型是实数\*}

then n := n + 1 {\*对应的函数标号+1\*}

end;

{ odd, chr } 4,5: ts := [ints]; {\*如果是4,5号操作,那么完成判奇和ascii码转化成字符的操作,要求传入的是脏呢挂车能\*}

{ odr } 6: ts := [ints,bools,chars]; {\*6号操作允许类型是整型,布尔型或者字符型\*}

{ succ,pred } 7,8 : begin {对于7,8号操作}

ts := [ints, bools,chars]; {\*允许参数类型是整型,布尔型或者字符型\*}

tab[i].typ := x.typ {\*记录类型\*}

end;

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

{ sin,cos,... } begin

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

if x.typ = ints {\*如果为整型\*}

then emit1(26,0) {\*先将整型转成实型\*}

end;

end; { case }

if x.typ in ts {\*如果函数的类型符合要求的符号集\*}

then emit1(8,n) {\*调用8号指令,生成标准函数\*}

else if x.typ <> notyp {\*如果x的类型未定义\*}

then error(48); {\*报48号错误,类型错误\*}

end

else begin { n in [17,18] } {\*如果编号是17或者18,即判断输入是否结束\*}

if sy <> ident {\*传入的首先应当是标识符\*}

then error(2) {\*不是标识符报错\*}

else if id <> 'input ' {\*如果对应的id不是'input '\*}

then error(0) {\*报0号错误,未知id\*}

else insymbol; {\*没错的话读取下一个sym\*}

emit1(8,n); {\*生成标准函数\*}

end;

x.typ := tab[i].typ; {\*记录返回值类型\*}

if sy = rparent {\*识别是否遇到右括号\*}

then insymbol {\*获取下一个sym,标准函数处理过程结束\*}

else error(4) {\*如果没有识别到右括号,报4号错误\*}

end { standfct } ;

begin { factor } {\*因子分析程序开始\*}

x.typ := notyp; {\*初始化返回值类型\*}

x.ref := 0; {\*初始化返回的位置指针\*}

test( facbegsys, fsys,58 ); {\*检查当前的符号是否是合法的因子开始符号\*

while sy in facbegsys do {\*当当前的符号是因子的开始符号时\*}

begin

if sy = ident {\*如果识别到标识符}

then begin

i := loc(id); {\*获取当前标识符在符号表中的位置保存到i\*}

insymbol; {\*获取下一个sym\*}

with tab[i] do {\*对当前符号对应的表项进行操作\*}

case obj of {\*对于不同的obj属性执行不同的操作\*}

konstant: begin {\*如果是常量类型\*}

x.typ := typ; {\*返回值的类型就设置为表中记录的typ\*}

x.ref := 0; {\*索引值设置为0\*}

if x.typ = reals {\*如果是实数类型的常量\*}

then emit1(25,adr) {\*将实数装入数据栈,注意实数常量的adr对应着其在rconst实常量表中的位置\*}

else emit1(24,adr) {\*如果是整型直接存入栈顶即可\*}

end;

vvariable:begin {\*如果换成变量类型\*}

x.typ := typ; {\*获得需要返回类型\*}

x.ref := ref; {\*获得需要返回地址\*}

if sy in [lbrack, lparent,period] {\*如果标识符后面跟的是左方括号,左括号或者是句号,说明该变量存在子结构\*}

then begin

if normal {\*如果是实形参\*}

then f := 0 {\*取地址\*}

else f := 1; {\*否则是变量形参,取值并放到栈顶\*}

emit2(f,lev,adr); {\*生成对应的代码\*}

selector(fsys,x); {\*处理子结构\*}

if x.typ in stantyps {\*如果是标准类型\*}

then emit(34) {\*将该值放到栈顶\*}

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 {\*如果是值形参\*}

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; {\*如果既不是标准类型又不是值形参,执行取值操作\*}

emit2(f,lev,adr) {\*生成对应指令\*}

end

end;

typel,prozedure: error(44); {\*如果是类型类型或者过程类型,报44号类型错误\*}

funktion: begin {\*如果是函数符号\*}

x.typ := typ; {\*记录类型\*}

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) {\*如果层次为零,调用标准函数\*}

end

end { case,with }

end

else if sy in [ charcon,intcon,realcon ] {\*如果符号的类型是字符类型,整数类型或者实数类型\*}

then begin

if sy = realcon {对于实数类型\*}

then begin

x.typ := reals; {\*将返回的type设置为实型\*}

enterreal(rnum); {\*将该实数放入实数表,rnum存有实数的值\*}

emit1(25,c1) {\*将实常量表中第c1个(也就是刚刚放进去的)元素放入栈顶\*}

end

else begin

if sy = charcon {\*对于字符类型\*}

then x.typ := chars {\*记录返回的类型是字符型\*}

else x.typ := ints; {\*否则肯定是整形啦,要不进不来这个分支\*}

emit1(24,inum) {\*装入字面变量,可以看出字符型装的是ascii码值\*}

end;

x.ref := 0; {\*返回的ref设置为0\*}

insymbol {\*获取下一个sym\*}

end

else if sy = lparent {\*如果符号的类型是左括号\*}

then begin

insymbol; {\*获取下一个sym\*}

expression(fsys + [rparent],x); {\*调用处理表达式的递归子程序处理括号中的表达式\*}

if sy = rparent {\*如果遇到了右括号\*}

then insymbol {\*获取下一个sym\*}

else error(4) {\*没有右括号报4号错误\*}

end

else if sy = notsy {\*如果符号的类型未定义\*}

then begin

insymbol; {\*获取下一个sym\*}

factor(fsys,x); {\*递归调用因子的分析子程序\*}

if x.typ = bools {\*如果返回的类型是布尔型\*}

then emit(35) {\*生成逻辑非指令\*}

else if x.typ <> notyp {\*如果因子的类型依旧未定义\*}

then error(32) {\*生成32指令,退出过程\*}

end;

test(fsys,facbegsys,6) {\*检查当前符号是否合法\*}

end { while }

end { factor };

begin { term } {\*开始处理项(term)\*}

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; {\*运算符是sy所代表的类型\*}

insymbol; {\*获取下一个sym\*}

factor(fsys+[times,rdiv,idiv,imod,andsy],y ); {\*继续调用因子分析程序来分析因子,获得第二个运算数存为y\*}

if op = times {\*如果遇到了乘号\*}

then begin

x.typ := resulttype(x.typ, y.typ); {\*求出计算之后结果的类型\*}

case x.typ of

notyp: ; {\*未定义类型不干事儿\*}

ints : emit(57); {\*整数生成整数乘指令\*}

reals: emit(60); {\*实数生成实数乘指令\*}

end

end

else if op = rdiv {\*除法运算\*}

then begin

if x.typ = ints

then begin

emit1(26,1); {\*整型转实型\*}

x.typ := reals;

end;

if y.typ = ints

then begin

emit1(26,0); {\*整型转实型\*}

y.typ := reals;

end;

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

then emit(61) {\*实型除法\*}

else begin

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

then error(33);

x.typ := notyp

end

end

else if op = andsy {\*与运算\*}

then begin

if( x.typ = bools )and(y.typ = bools) {\*必须两个运算数都是布尔类型\*}

then emit(56) {\*生成逻辑与运算\*}

else begin

if( x.typ <> notyp ) and (y.typ <> notyp) {\*类型不对报错,提示应该是布尔值\*}

then error(32);

x.typ := notyp

end

end

else begin { op in [idiv,imod] }

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

then if op = idiv {\*如果是除法\*}

then emit(58) {\*生成除法运算的代码\*}

else emit(59) {\*否则生成取模运算的代码\*}

else begin

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

then error(34); {\*类型出错报错\*}

x.typ := notyp

end

end

end { while }

end { term };

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

if sy in [plus,minus] {\*获得的是加减号\*}

then begin

op := sy; {\*记录运算符\*}

insymbol;

term( fsys+[plus,minus],x); {\*处理项\*}

if x.typ > reals {\*类型是 bools, chars, arrays, records\*}

then error(33) {\*由于不是算数运算类型,报错\*}

else if op = minus {\*如果是减号\*}

then emit(36) {\*去相反数\*}

end

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

while sy in [plus,minus,orsy] do

begin

op := sy;

insymbol;

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

if op = orsy {\*如果是or关键字\*}

then begin

if ( x.typ = bools )and(y.typ = bools) {\*操作数限定为bool\*}

then emit(51) {\*生成OR指令\*}

else begin

if( x.typ <> notyp) and (y.typ <> notyp) {\*类型不对报错\*}

then error(32);

x.typ := notyp

end

end

else begin

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

case x.typ of

notyp: ;

ints: if op = plus {\*整数加减\*}

then emit(52)

else emit(53);

reals:if op = plus {\*实数加减\*}

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] {\*判别多种数值比较符号\*}

then begin

op := sy;

insymbol;

simpleexpression(fsys,y); {\*获得第二个简单表达式的值\*}

if(x.typ in [notyp,ints,bools,chars]) and (x.typ = y.typ) {\*整型,布尔和字符都可以借用整型的运算\*}

then case op of {\*根据不同的符号来生成不同的PCODE\*}

eql: emit(45);

neq: emit(46);

lss: emit(47);

leq: emit(48);

gtr: emit(49);

geq: emit(50);

end

else begin

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) {\*对于实数同样生成不同的PCODE\*}

then case op of

eql: emit(39);

neq: emit(40);

lss: emit(41);

leq: emit(42);

gtr: emit(43);

geq: emit(44);

end

else error(35) {\*报35号错误\*}

end;

x.typ := bools

end

end { expression };

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

var x,y: item;

f : integer;

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

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

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

if tab[i].normal

then f := 0

else f := 1;

emit2(f,lv,ad);

if sy in [lbrack,lparent,period]

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

if sy = becomes {\*赋值符号\*}

then insymbol

else begin

error(51);

if sy = eql {\*等号容错\*}

then insymbol

end;

expression(fsys,y); {\*获得赋值符号右边的值\*}

if x.typ = y.typ

then if x.typ in stantyps

then emit(38) {\*完成赋值操作\*}

else if x.ref <> y.ref

then error(46)

else if x.typ = arrays {\*数组类型需要拷贝块\*}

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

else emit1(23,btab[x.ref].vsize) {\*拷贝btab中的记录项\*}

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

then begin

emit1(26,0);

emit(38)

end

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

then error(46) {\*报46号错误\*}

end { assignment };

procedure compoundstatement; {\*semicolon 处理过程\*}

begin

insymbol;

statement([semicolon,endsy]+fsys);

while sy in [semicolon]+statbegsys do

begin

if sy = semicolon

then insymbol

else error(14); {\*报14号错误\*}

statement([semicolon,endsy]+fsys)

end;

if sy = endsy

then insymbol

else error(57) {\*报51号错误\*}

end { compoundstatement };

procedure ifstatement; {\*if语句处理过程\*}

var x : item;

lc1,lc2: integer;

begin

insymbol;

expression( fsys+[thensy,dosy],x);

if not ( x.typ in [bools,notyp])

then error(17);{\*报17号错误\*}

lc1 := lc;

emit(11); { jmpc }

if sy = thensy

then insymbol

else begin

error(52);{\*报52号错误\*}

if sy = dosy

then insymbol

end;

statement( fsys+[elsesy]);

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;{\*case语句的处理过程\*}

var x : item;

i,j,k,lc1 : integer; {\*定义一系列临时变量\*}

casetab : array[1..csmax]of {\*csmax表示case个数的最大限度\*}

packed record

val,lc : index {\*index表示\*}

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 ); {\*因为标签都是常量,这里调用处理常量的过程来获得常量的值,存于lab\*}

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

then error(47) {\*报label类型错误\*}

else if i = csmax {\*如果可以声明的case达到了最大限度\*}

then fatal(6) {\*报6号严重错误,程序终止\*}

else begin

i := i+1; {\*移动case表的指针,声明新的case\*}

k := 0; {\*用来检查标号是否重复定义的变量\*}

casetab[i].val := lab.i; {\*保存新case的值\*}

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

repeat

k := k+1

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

if k < i {\*重复声明\*}

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

end

end { caselabel };

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

begin

if sy in constbegsys {\*确定当前符号是常量的类型集合\*}

then begin

caselabel; {\*获取一个标签\*}

while sy = comma do {\*如果有逗号说明是一个case对应多个标签的情况\*}

begin

insymbol; {\*继续获取标签的label\*}

caselabel {\*继续处理\*}

end;

if sy = colon {\*读到冒号,说明label声明结束了\*}

then insymbol {\*获取下一个sym\*}

else error(5); {\*没读到冒号,报5号错误\*}

statement([semicolon,endsy]+fsys); {\*递归调用statement来处理冒号之后需要执行的程序\*}

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

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

emit(10) {\*生成一条跳转指令来结束这一case分支\*}

end

end { onecase };

begin { casestatement }

insymbol; {\*获取下一个sym\*}

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); {\*报23号错误,case类型错误\*}

lc1 := lc; {\*记录当前PCODE代码的位置指针\*}

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

if sy = ofsy {\*如果接着读到了of关键字\*}

then insymbol {\*获取下一个sym\*}

else error(8); {\*丢失of关键字的情况报8号错\*}

onecase; {\*调用onecase方法处理\*}

while sy = semicolon do {\*遇到了分号,说明还有更多的case分支\*}

begin

insymbol; {\*获取下一个sym\*}

onecase {\*处理下一个sym\*}

end;

code[lc1].y := lc; {\*此时确定了情况表的开始地址,回填给之前声明的SWT代码,确保其能够成功跳转\*}

for k := 1 to i do {\*便利所有case分支\*}

begin {\*建立情况表\*}

emit1( 13,casetab[k].val); {\*建立查找的值\*}

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

end;

emit1(10,0); {\*生成JMP代码,说明情况表结束\*}

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

code[exittab[k]].y := lc; {\*现在的lc指向情况表结束之后的位置,将各分支的结束跳转地址指向这里\*}

if sy = endsy {\*如果遇到了end关键字\*}

then insymbol {\*读取下一个sym,case处理完毕\*}

else error(57) {\*否则报57号错误\*}

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; {\*获取下一个sym\*}

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 {\*如果确实是分号\*}

then insymbol {\*获取下一个sym\*}

else error(14); {\*报14号错,提示分号错误\*}

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

end;

if sy = untilsy {\*如果遇到了until关键字\*}

then begin

insymbol; {\*获取下一个sym,即循环条件\*}

expression(fsys,x); {\*处理该表达式,获得其类型\*}

if not(x.typ in [bools,notyp] ) {\*如果不是未定义类型或者布尔型的表达式\*}

then error(17); {\*报17号错误,提示需要布尔型表达式\*}

emit1(11,lc1); {\*生成一条条件跳转指令,如果表达式的值是假的,则跳转回repeat开始的位置重新执行一遍\*}

end

else error(53) {\*没找到until,报53号错\*}

end { repeatstatement };

procedure whilestatement; {\*处理while循环的过程\*}

var x : item;

lc1,lc2 : integer;

begin

insymbol;

lc1 := lc;

expression( fsys+[dosy],x);

if not( x.typ in [bools, notyp] )

then error(17); {\*报17号错误\*}

lc2 := lc;

emit(11);

if sy = dosy

then insymbol

else error(54); {\*报54错误\*}

statement(fsys);

emit1(10,lc1);

code[lc2].y := lc

end { whilestatement };

procedure forstatement; {\*处理for循环语句\*}

var cvt : types;

x : item;

i,f,lc1,lc2 : integer;

begin

insymbol; {\*获取下一个sym\*}

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,实际上是计数变量\*}

insymbol; {\*获取下一个sym\*}

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\*}

then error(37) {\*报37号错\*}

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\*}

then error(18) {\*报18号错误\*}

end

else begin {\*如果符号的类型也不是变量\*}

error(37); {\*报37号错误\*}

cvt := ints {\*将计数变量类型设置为整型\*}

end

end

else skip([becomes,tosy,downtosy,dosy]+fsys,2); {\*跳过无用符号\*}

if sy = becomes {\*如果识别到了赋值符号\*}

then begin

insymbol; {\*获取下一个sym\*}

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\*}

then error(19); {\*报19号错误\*}

end

else skip([tosy, downtosy,dosy]+fsys,51); {\*未识别到赋值符号,则继续执行\*}

f := 14; {\*生成指令的编号,暂存14号\*}

if sy in [tosy,downtosy] {\*如果当前符号是to关键字或者downto关键字,其中to是每次循环变量自加一,downto是每次循环变量自减一\*}

then begin

if sy = downtosy {\*如果是down\*}

then f := 16;

insymbol; {\*获取下一个sym\*}

expression([dosy]+fsys,x); {\*调用处理表达式的递归子程序处理括号中的表达式\*}

if x.typ <> cvt {\*如果表达式的类型和左边的计数变量不同\*}

then error(19) {\*报19号错误\*}

end

else skip([dosy]+fsys,55); {\*跳过直到do之前的代码段\*}

lc1 := lc; {\*记录下句F1U指令的位置\*}

emit(f); {\*生成F1U或F1D指令,进行循环体的入口测试\*}

if sy = dosy {\*如果当前符号是do关键字\*}

then insymbol {\*获取下一个sym\*}

else error(54); {\*没找到do,报54号错误\*}

lc2 := lc; {\*获取循环体开始代码的位置\*}

statement(fsys); {\*递归调用statement来处理循环体语句\*}

emit1(f+1,lc2); {\*结束时生成F2U或F2D指令\*}

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

end { forstatement };

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

var i,f : integer;

x,y : item;

begin

case n of

1,2 : begin { read }

if not iflag

then begin

error(20); {\*报20号错误\*}

iflag := true

end;

if sy = lparent

then begin

repeat

insymbol;

if sy <> ident

then error(2) {\*报2号错误\*}

else begin

i := loc(id);

insymbol;

if i <> 0

then if tab[i].obj <> vvariable

then error(37) {\*报37号错误\*}

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) {\*报41号错误\*}

end

end;

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

until sy <> comma; {\*直到符号是comma\*}

if sy = rparent

then insymbol {\*获取符号\*}

else error(4) {\*报4号错误\*}

end;

if n = 2

then emit(62)

end;

3,4 : begin { write }

if sy = lparent

then begin

repeat

insymbol;

if sy = stringcon

then begin

emit1(24,sleng);

emit1(28,inum);

insymbol

end

else begin

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

if not( x.typ in stantyps )

then error(41); {\*报41号错误\*}

if sy = colon {\*若符号是colon\*}

then begin

insymbol; {\*获取下一个符号\*}

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

if y.typ <> ints

then error(43); {\*报43号错误\*}

if sy = colon

then begin

if x.typ <> reals

then error(42); {\*报42号错误\*}

insymbol;

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

if y.typ <> ints

then error(43); {\*报43号错误\*}

emit(37)

end

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

end

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

end

until sy <> comma; {\*直到符号是comma\*}

if sy = rparent

then insymbol {\*获取符号\*}

else error(4) {\*报4号错误\*}

end;

if n = 4

then emit(63)

end; { write }

end { case };

end { standproc } ;

begin { statement }

if sy in statbegsys+[ident]

then case sy of

ident : begin

i := loc(id);

insymbol;

if i <> 0

then case tab[i].obj of

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;

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; {\*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\*}

then fatal(5); {\*报5号错误\*}

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14号错误\*}

enterblock;

prb := b;

display[level] := b;

tab[prt].typ := notyp;

tab[prt].ref := prb;

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

then parameterlist;

btab[prb].lastpar := t;

btab[prb].psize := dx;

if isfun

then if sy = colon

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 )

end

else error(5); {\*报5号错误\*}

if sy = semicolon {如果符号是 semicolon}

then insymbol {\*获取下一个符号\*}

else error(14); {\*报14号错误\*}

repeat

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;

tab[prt].adr := lc;

insymbol;

statement([semicolon,endsy]+fsys);

while sy in [semicolon]+statbegsys do

begin

if sy = semicolon

then insymbol {\*获取下个符号\*}

else error(14); {\*报14号错误\*}

statement([semicolon,endsy]+fsys);

end;

if sy = endsy {\*若符号是 endsy\*}

then insymbol {\*获取下个符号\*}

else error(57); {\*报57号错误\*}

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

end { block };

procedure interpret;

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

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 } {\*基址地址\*}

h1,h2,h3: integer; {\*临时变量\*}

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

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

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

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

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

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;

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);

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 }

{\*以下为不同PCODE所对应的操作\*}

procedure inter0;

begin

case ir.f of

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

t := t + 1; {\*栈顶指针上移\*}

if t > stacksize {\*如果超过了栈的大小上限\*}

then ps := stkchk {\*将ps设置为stkchk,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

then ps := stkchk

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

end;

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

h1 := ir.y;

h2 := ir.x;

h3 := b;

repeat

display[h1] := h3;

h1 := h1-1; {level-1} {\*层次减 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

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

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为计数变量的值自增一\*}

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;

if h1 >= s[t].i

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

else begin

pc := ir.y;

t := t - 3

end

end;

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

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

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

if h1 >= s[t].i

then begin

s[h2].i := h1;

pc := ir.y

end

else t := t-3;

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; {\*DL的值\*}

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

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

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

for h3 := t+1 to h4 do

s[h3].i := 0;

b := h1;

t := h4;

pc := tab[h2].adr;

if stackdump

then dump

end;

end { case }

end; { inter1 }

procedure inter2;

begin

case ir.f of

20 : begin { index1 } {\*索引\*}

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 } {\*索引\*}

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

h2 := atab[h1].low;

h3 := s[t].i;

if h3 < h2

then ps := inxchk

else if h3 > atab[h1].high

then ps := inxchk

else begin

t := t-1;

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

end

end;

22 : begin { load block } {\*装入块,LDB\*}

h1 := s[t].i; {\*获取栈顶值\*}

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 }

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 } {\*装入字面变量,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 } {\*字符串， 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 }

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 } {\*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,注意要保留函数名\*}

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];

35 : s[t].b := not s[t].b; {\*逻辑非运算,将栈顶布尔值取反,NOT\*}

36 : s[t].i := -s[t].i; {\*取整数的相反数操作,MUS\*}

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 } {\*将栈顶内容存入以次栈顶为地址的单元,STO\*}

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

t := t-2

end;

39 : begin {\*实数相等,EQR\*}

t := t-1;

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

end;

end { case }

end; { inter3 }

procedure inter4;

begin

case ir.f of

40 : begin {\*实数不等,NER\*}

t := t-1;

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

end;

41 : begin {\*实数小于,LSR\*}

t := t-1;

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

end;

42 : begin {\*实数小于等于,LER\*}

t := t-1;

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

end;

43 : begin {\*数大于,GTR\*}

t := t-1;

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

end;

44 : begin {\*实数大于等于,GER\*}

t := t-1;

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

end;

45 : begin {\*整数相等,EQL\*}

t := t-1;

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

end;

46 : begin {\*整型不等,NEQ\*}

t := t-1;

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

end;

47 : begin {\*整型小于,LSS\*}

t := t-1;

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

end;

48 : begin {\*整型小于等于,LEQ\*}

t := t-1;

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

end;

49 : begin {\*整型大于,GRT\*}

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 {\*整型大于等于,GEQ\*}

t := t-1;

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

end;

51 : begin {\*OR指令,ORR\*}

t := t-1;

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

end;

52 : begin {\*整数加,ADD\*}

t := t-1;

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

end;

53 : begin {\*整数减,SUB\*}

t := t-1;

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

end;

54 : begin {\*实数加,ADR\*}

t := t-1;

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

end;

55 : begin {\*实数减,SUR\*}

t := t-1;

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

end;

56 : begin {\*与运算,AND\*}

t := t-1;

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

end;

57 : begin {\*整数乘,MUL\*}

t := t-1;

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

end;

58 : begin {\*整数除法,DIV\*}

t := t-1;

if s[t+1].i = 0

then ps := divchk

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

end;

59 : begin {\*取模运算,MOD\*}

t := t-1;

if s[t+1].i = 0

then ps := divchk

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

end;

end { case }

end; { inter5 }

procedure inter6;

begin

case ir.f of

60 : begin {\*实数乘\*}

t := t-1;

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

end;

61 : begin {\*实数除\*}

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 {\*根据不同的错误信息来进行报错\*}

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

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

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

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

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

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

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

end;

h1 := b;

blkcnt := 10; { post mortem dump }

repeat

writeln( prr );

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; {\*程序运行前的准备过程\*}

begin

key[1] := 'and '; {\*定义一系列保留字\*}

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; {\*定义保留字对应的符号\*}

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; {\*定义特殊字符对应的sym\*}

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; {\*这个过程负责将全部标准类型的信息登陆到table中\*}

begin

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

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

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

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

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

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

enter('integer ',typel,ints,1);

enter('abs ',funktion,reals,0);

enter('sqr ',funktion,reals,2);

enter('odd ',funktion,bools,4);

enter('chr ',funktion,chars,5);

enter('ord ',funktion,ints,6);

enter('succ ',funktion,chars,7);

enter('pred ',funktion,chars,8);

enter('round ',funktion,ints,9);

enter('trunc ',funktion,ints,10);

enter('sin ',funktion,reals,11);

enter('cos ',funktion,reals,12);

enter('exp ',funktion,reals,13);

enter('ln ',funktion,reals,14);

enter('sqrt ',funktion,reals,15);

enter('arctan ',funktion,reals,16);

enter('eof ',funktion,bools,17);

enter('eoln ',funktion,bools,18);

enter('read ',prozedure,notyp,1);

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

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

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

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

end;

begin { main }

setup; {\*初始化变量\*}

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 ];

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 ?'); {\*代码输入文件\*}

readln( inf );

assign( psin, inf );

reset( psin );

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

readln( outf );

assign( psout, outf );

rewrite( psout );

assign ( prd, 'con' );

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

readln( fprr );

assign( prr, fprr );

reset ( prd );

rewrite( prr );

t := -1; {\*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\*}

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,不是的话就报错\*}

then error(3)

else begin

insymbol; {\*获取下一个符号\*}

if sy <> ident {\*应该是程序名,不是则报错\*}

then error(2)

else begin

progname := id;

insymbol;

if sy <> lparent

then error(9)

else repeat

insymbol; {\*获取下一个符号\*}

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 {\*获取下一个符号\*}

else error(4); {\*报4号错误\*}

if not oflag then error(20) {\*报20号错误\*}

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 {\*若符号不是period\*}

then error(2); {\*报2号错误\*}

emit(31); { halt }

if prtables

then printtables;

if errs = []

then interpret

else begin

writeln( psout );

writeln( psout, 'compiled with errors' ); {\*报错提示\*}

writeln( psout );

errormsg;

end;

writeln( psout );

close( psout );

close( prr )

end.