

**编译技术课程设计**

实验报告

北京航空航天大学

计算机学院

陈麒先

16061160

二○一八年十一月

**郑重声明**

**关于诚实守信公约：**

**本实验报告中参考了文献或互联网资料的部分均有引用标注，另有与老师或同学研究后的成果引用将在特别致谢中说明，其余部分均为本人通过查阅课件、课堂笔记和教材后独立思考的结果。另，实验报告撰写仓促，如有错误，在所难免，欢迎批评指正！特此声明。**

**陈麒先**

**原创性声明**

**本实验报告未注释部分内容由作者原创。抄袭行为在任何情况下都是不能容忍的(COPY is strictly prohibited under any circumstances)！转载或引用须征得作者本人同意，并注明出处！勿谓言之不预！**

**陈麒先**



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 关键字总数 }

alng = 10; { no. of significant chars in identifiers 标识符长度10 }

llng = 121; { input line length }

emax = 322; { max exponent of real numbers 实型指数位上限 }

emin = -292; { min exponent 实型指数位下限 }

kmax = 15; { max no. of significant digits 数字位数上限 }

tmax = 100; { size of table 符号表大小 }

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

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

c2max = 20; { size of real constant table 实常量表大小 }

csmax = 30; { max no. of cases case的最大数量}

cmax = 800; { size of code }

lmax = 7; { maximum level 最大嵌套层次深度为7 }

smax = 600; { size of string-table }

ermax = 58; { max error no. 共有58种错误类型 }

omax = 63; { highest order code 指令种类总数63 }

xmax = 32767; { 2\*\*15-1 x index 索引值边界 }

nmax = 32767; { 2\*\*15-1 数值范围 }

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);{\*符号种类\*}

index = -xmax..+xmax;{\*定义index类型 数组下标的范围[-xnax,xmax]\*}

alfa = packed array[1..alng]of char; {\*定义alpha类型 长度为10的字符数组\*}

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

{\*定义objecttyp类型 常量 变量 类型 过程 函数\*}

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

{\*定义types类型 无类型（过程名）整型 实型 布尔型 字符型 数组型 记录型\*}

symset = set of symbol;{\*符号集\*}

typset = set of types; {\*类型集\*}

item = record {\*项 包含类型和位置\*}

typ: types;

ref: index;

end;

order = packed record {\*p-code指令类型\*}

f: -omax..+omax; {\*名\*}

x: -lmax..+lmax; {\*层深\*}

y: -nmax..+nmax {\*数值\*}

end;

var ch: char; { last character read from source program 预读的字符 }

rnum: real; { real number from insymbol 读入的实数 }

inum: integer; { integer from insymbol [多重含义]读字符读到的数字/该字符串在stab中起始位置 }

sleng: integer; { string length 该字符串长度 }

cc: integer; { character counter 字符计数 }

lc: integer; { program location counter P代码表索引变量 }

ll: integer; { length of current line 当前行长度 }

errpos: integer; { 错误发生位置 }

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

{\*t 符号表tab的索引变量；b 分程序表btab的索引变量\*}

{\*a 数组表atab的索引变量；sx 字符串表stab的索引变量\*}

{\*c1 ；c2 实常量表rconst的索引变量\*}

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

{\*prtables打印表标志位 stackdump是否导出的标志位\*}

{\*各种flag标志位\*}

sy: symbol; { last symbol read by insymbol 预读的符号类型码 }

errs: set of 0..ermax; {\*错误信息数组\*}

id: alfa; { identifier from insymbol 读到的标识符 }

progname: alfa; {\*程序名\*}

stantyps: typset; {\*标准类型\*}

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

{\*各种首符号集的声明\*}

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

key: array[1..nkw] of alfa;

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

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

{\*该用法类似字典 索引下标为字符型 特殊符号数组\*}

display: array[0..lmax] of integer;{\*分程序索引表 表项即指向btab指针\*}

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

packed record { 用法packed record 类似字典列表 }

name: alfa; {\*标识符名前10位\*}

link: index;{\*同一分程序上一个标识符在符号表中的位置\*}

obj: objecttyp;{\*标识符种类\*}

typ: types;{\*标识符类型\*}

ref: index;{\*指针值 分别在其他表中的登录位置\*}

normal: boolean;{\*只有变量形参标识符填false其余填true\*}

lev: 0..lmax;{\*标识符所在层次\*}

adr: integer {\*地址 与具体的符号类型有关\*}

end;

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

packed record

inxtyp,eltyp: types;

{\*inxtyp索引类型 可选ints/bools/chars\*}

{\*eltye元素类型 类型任意\*}

elref,low,high,elsize,size: index

{\*low/high数组上下界 elsize/size数组元素大小和数组大小\*}

end;

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

packed record

last, lastpar, psize, vsize: index

{\*last 当前标识符在tab中的位置\*}

{\*lastpar 过程/函数最后一个参数在tab中的位置\*}

{\*psize 参数在运行栈中所占存储单元数\*}

{\*vsize 局部变量在运行栈中所占存储单元数\*}

end;

stab: packed array[0..smax] of char; { string table }

{\*每个字符串存在stab 起始位置放在inum 长度放在sleng\*}

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

code: array[0..cmax] of order; {\*P代码表\*}

{\*编译出的目标指令放入code 解释执行从其中一条条取出执行\*}

psin,psout,prr,prd:text; { default in pascal p 默认值 }

{\*psin为输入文件 psout为打印输出文件\*}

inf, outf, fprr: string;

procedure errormsg;

{\*错误信息提示\*}

var k : integer;

msg: array[0..ermax] of alfa; {\*一共有58种错误类型提示\*}

{\*以下开始对错误信息字符串数组赋值\*}

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); {\*输出一个空行\*}

writeln(psout,'key words'); {\*该用法为向输出文件格式输出语句\*}

k := 0;

{\*两层循环输出错误信息\*}

while errs <> [] do {\*循环到错误信息数组为空\*}

begin

while not( k in errs )do k := k + 1; {\*循环k递增直到errs中包含编号k的错误信息\*}

writeln(psout, k, ' ', msg[k] ); {\*向psout输出错误信息\*}

errs := errs - [k] {\*更新错误信息数组出去k类型错误\*}

end { while errs }

end { errormsg } ;

procedure endskip;

{\*在被跳过的源程序代码下面加下划线\*}

begin { underline skipped part of input }

while errpos < cc do

begin

write( psout, '-');

errpos := errpos + 1

end;

skipflag := false {\*不再继续跳读\*}

end { endskip };

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

{\*读下一字符\*}

begin

{\*先对读到行尾情况进行判断\*}

if cc = ll {\*读到行尾\*}

then begin{\*000000\*}

if eof( psin ) {\*如果是程序尾\*}

then begin

writeln( psout );

writeln( psout, 'program incomplete' );

errormsg; {\*程序不完整错误\*}

exit;

end;

{\*以下进行 行出错处理\*}

if errpos <> 0 {\*错误位置不为初始值 即该行有错误\*}

then begin

if skipflag then endskip; {\*如果跳读标志位为真 则在跳读部分加下划线\*}

writeln( psout ); {\*输出空行\*}

errpos := 0 {\*重置错误位置为初值\*}

end;

write( psout, lc: 5, ' '); {\*输出出错位置\*}

ll := 0; {\*重置行长度为初值\*}

cc := 0; {\*重置读字符位置为初值\*}

while not eoln( psin ) do

{\*因为在读字符之前当前行已经读完了，

而且psin的指针指向的是下一行代码,

于是进行循环将psin所在行的代码装入line中\*}

begin {\*输入代码文件的最后一行（前面有判断保证）\*}

ll := ll + 1; {\*循环按字符读取下一行到line\*}

read( psin, ch );

write( psout, ch );

line[ll] := ch

end;

ll := ll + 1;

readln( psin ); {\*让psin读下一行，以保证下次调用pisn的时候是下一行指针\*}

line[ll] := ' '; {\*最后用空格作为行结尾\*}

writeln( psout );

end;{\*判断读到行尾的特殊情况处理结束\*}

cc := cc + 1; {\*正常读取字符\*}

ch := line[cc];{\*从line中读出下一个字符\*}

end { nextch };

procedure error( n: integer );

{\*打印出错位置和出错编号, 并将错误编号加入errs中\*}

{\*传入参数为错误类型码\*}

begin

if errpos = 0

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

if cc > errpos {\*当前错误在上次出错位置之后才输出错误信息，避免重复输出相同错误\*}

then begin

write( psout, ' ': cc-errpos, '^', n:2); {\*打印出若干空格使箭头指向出错位置\*}

errpos := cc + 3;

errs := errs +[n] {\*将错误编号加入errs中\*}

end

end { error };

procedure fatal( n: integer );

{\*打印表格溢出信息并终止汇编程序\*}

{\*传入参数为溢出表的类型编码\*}

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}

{\*读取下一符号 并能处理注释行\*}

label 1,2,3; {\*定义label,用于后面go to的跳转\*}

var i,j,k,e: integer;

{\*i，j分别为二分查找头尾 k为读字符下标 e为实数指数\*}

procedure readscale;

{\*处理实数的指数部分\*}

var s,sign: integer;{\*sign 记录符号位+1为正-1为负 s为数值\*}

begin

nextch;

sign := 1;

s := 0;

if ch = '+'

then nextch

else if ch = '-'

then begin

nextch;

sign := -1

end;

if not(( ch >= '0' )and (ch <= '9' )) {\*读完符号以后读到的不是数字 错误\*}

then error( 40 )

else repeat

s := 10\*s + ord( ord(ch)-ord('0')); {\*字符串转数字 存入s\*}

nextch;

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

e := s\*sign + e {\*上述读到的数字加入到e中\*}

end { readscale };

procedure adjustscale;

{\*根据小数位数和指数大小求出实数数值\*}

var s : integer;

d, t : real;

begin

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

then error(21)

else if k + e < emin

then rnum := 0 {\*小于精度下界记该实数为0\*}

else begin

s := abs(e);

t := 1.0; {\*临时值\*}

d := 10.0; {\*十进制的基\*}

repeat

while not odd(s) do {\*为了加快收敛速度采用折半法\*}

begin

s := s div 2; {\*把幂次中2的倍数提出来\*}

d := sqr(d) {\*乘到平方项中 sqr为d平方\*}

end;

s := s - 1; {\*减一之后变成偶数\*}

t := d \* t {\*t自乘一次,相当于幂次加一，t此时为10的e次方\*}

until s = 0;

if e >= 0 {\*根据指数的正负决定实数是乘除该位数值\*}

then rnum := rnum \* t

else rnum := rnum / t

end

end { adjustscale };

procedure options;

{\*处理编译时的可选项\*}

procedure switch( var b: boolean );

{\*处理编译可选项中的“+”,“-”号\*}

begin

b := ch = '+'; {\*此处是修改传入的prtables/stackdump标志位 读到+则输出\*}

if not b

then if not( ch = '-' ) {\*既非+也非- 则错误符号\*}

then begin { print error message }

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

nextch; {\*读到【\*/,】停止 即在遇到非+-符号之后跳读无用符号\*}

end

else nextch {\*读下一符号\*}

else nextch

end { switch };

begin { options }

repeat

nextch;

if ch <> '\*'

then begin

if ch = 't' {\*编译选项 t打印表 s导出栈\*}

then begin

nextch;

switch( prtables )

end

else if ch = 's'

then begin

nextch;

switch( stackdump )

end;

end

until ch <> ','

end { options };

begin { insymbol }

{\*以下开始获取字符\*}

{\*注意此处第一个标签开始\*}

1: while( ch = ' ' ) or ( ch = chr(9) ) do {\*跳过所有的空格和\t\*}

nextch; { space & htab }

case ch of

{\*ch读到了字母\*}

'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 {\*字符类型分支开始\*}

k := 0; {\*当前读字符的下标\*}

id := ' '; {\*当前读到的标识符（10位空串）\*}

repeat

if k < alng {\*alng是限定的标识符长度\*}

then begin

k := k + 1;

id[k] := ch {\*读id项\*}

end;

nextch

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

{\*直到不是标识符成分为止\*}

{\*以下二分查找该标识符是否为关键字\*}

i := 1; {\*二分头\*}

j := nkw; {\*二分查找尾\*}

repeat

k := ( i + j ) div 2;

if id <= key[k]

then j := k - 1;

if id >= key[k]

then i := k + 1;

until i > j;

if i - 1 > j

then sy := ksy[k] {\*如果在key表中查到则符号为关键字值\*}

else sy := ident {\*查找失败则为标识符\*}

end {\*end of wordsymbol\*};

{\*ch读到数字字符\*}

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

begin {\*数值型分支开始\*}

k := 0;

inum := 0;

sy := intcon; {\*设置sy为intcon表示数字常量\*}

repeat

inum := inum \* 10 + ord(ch) - ord('0'); {\*字符转数字串 更新当前总数值inum\*}

k := k + 1;

nextch {\*读字符\*}

until not (( ch >= '0' ) and ( ch <= '9' )); {\*循环读直到非数字为止\*}

if( k > kmax ) or ( inum > nmax )

then begin {\*超出数值范围限制错误 数字和读数字下标回零\*}

error(21);

inum := 0;

k := 0

end;

if ch = '.' {\*读到小数点\*}

then begin

nextch;

if ch = '.' {\*连读到两个小数点？\*}

then ch := ':' {\*认为该符号为：\*}

else begin

sy := realcon; {\*该符号存为实常数类型\*}

rnum := inum; {\*实数赋值为读到的整数部分\*}

e := 0; {\*指数\*}

{\*以下循环读小数\*}

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

begin

e := e - 1;

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

{\*把小数部分先当做整数来存\*}

nextch

end;

if e = 0

then error(40); {\*小数点后读到的不是数字错误\*}

if ch = 'e' {\*如果下一字符为e\*}

then readscale;{\*则后面为指数 应调用处理实数的指数部分子程序\*}

if e <> 0 then adjustscale {\*根据小数位数和指数大小求出实数数值\*}

end

end

else if ch = 'e'

{\*如果读到小数点之前就读到e 说明前面为一个整数\*}

then begin

sy := realcon; {\*符号是实型\*}

rnum := inum;

e := 0;

readscale; {\*指数处理\*}

if e <> 0

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

end;

end; {\*数字处理结束\*}

':':

{\*如果ch读到了:\*}

begin

nextch;

if ch = '='

then begin

sy := becomes; {\*becomes 为赋值:= \*}

nextch

end

else sy := colon {\*只是一个:\*}

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; {\*..算作和冒号一类,处理array[1..max]一类\*}

nextch

end

else sy := period {\*单个.认为是句点（程序段结尾符）\*}

end;

'''': {\*是单引号\*}

begin

k := 0;

{\*标签2 跳转返回入口\*}

2: nextch;

if ch = ''''

then begin

nextch;

if ch <> '''' {\*如果不是两个连续的单引号\*}

{\*否则3个连续单引号表示1个单引号字符 应送stab中\*}

then goto 3 {\*读到了字符串尾端的单引号 进入3保存该字符串\*}

end;

{\*只读到1个单引号之后就读到其他字符 被认为是字符串中的内容\*}

if sx + k = smax {\*sx为stab的索引变量\*}{\*字符串表溢出\*}

then fatal(7);

stab[sx+k] := ch; {\*把读到的字符存入字符表中\*}

k := k + 1;

if cc = 1 {\*因为在nextch中当cc=ll后下一行开始cc=1\*}

then begin { end of line } {\*字符串不允许换行 否则k赋0 进入3 表示出错\*}

k := 0;

end

else goto 2;

3: if k = 1 {\*引号之间只有一个字符\*}

then begin

sy := charcon; {\*sym类型为字符类型\*}

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

end

else if k = 0

then begin {\*双引号中间未读到字符异常\*}

error(38);

sy := charcon; {\*sym类型为空字符类型\*}

inum := 0 {\*inum存储该字符的ASCII值存成\0\*}

end

else begin

sy := stringcon; {\*双引号之间为字符串类型\*}

inum := sx; {\*字符串类型inum为该字符串在stab中起始位置\*}

sleng := k; {\*字符串长度\*}

sx := sx + k {\*sx在此更新（由此可见stab上各个字符串连续存储）\*}

end

end;

'(':

{\*左括号：注释和编译选项的处理\*}

begin

nextch;

if ch <> '\*'

then sy := lparent {\*若下一字符不是\*则该符号是一个左括号\*}

else begin { comment } {\*否则是一个注释\*}

nextch;

if ch = '$' {\*(\*$开头的为编译选项 进入编译选项处理子程序\*}

then options;

repeat

while ch <> '\*' do nextch; {\*直到读到下一个\*为止\*}

nextch

until ch = ')'; {\*直到读到)为止\*}

nextch;

goto 1 {\*跳读 回到读符号程序头\*}

end

end;

'{':

{\*注释行的跳读\*}

begin

nextch;

if ch = '$' {\*{$是编译选项设置\*}

then options; {\*进入编译选项处理子程序\*}

while ch <> '}' do {\*注释行的跳读指导读到注释尾}\*}

nextch;

nextch;

goto 1 {\*回到读符号开始\*}

end;

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

{\*读到运算符\*}

begin

sy := sps[ch]; {\*符号赋予运算符类型\*}

nextch

end;

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

{\*读到其他非法字符\*}

begin

error(24);

nextch;

goto 1 {\*实现了非法字符的跳读功能 回到读字符子程序开始\*}

end

end { case }

end { insymbol };

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

{\*(分程序外使用):把标准类型、过程和函数的名字登录到符号表\*}

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; {\*主程序开始之前就登录了一些自带的,所以级别最高\*}

adr := x3;

end

end; { enter }

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

{\*登录数组信息向量表atab\*}

{\*参数tp为数组类型 lh为数组上下界\*}

begin

if l > h {\*上下界错误处理\*}

then error(27);

if( abs(l) > xmax ) or ( abs(h) > xmax ) {\*上下界错误处理\*}

then begin

error(27);

l := 0;

h := 0;

end;

if a = amax {\*数组索引溢出\*}

then fatal(4) {\*输出表溢出信息\*}

else begin {\*正常情况 填表\*}

a := a + 1;

with atab[a] do

begin

inxtyp := tp;

low := l;

high := h

end

end

end { enterarray };

procedure enterblock;

{\*登录分程序表btab\*}

begin

if b = bmax {\*分程序表溢出\*}

then fatal(2)

else begin

b := b + 1;

btab[b].last := 0; {\*指向分程序中最后一个标识符在tab表中的位置\*}

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

end

end { enterblock };

procedure enterreal( x: real );

{\*登录实常量表rconst\*}

{\*参数x是待登录表的实数值\*}

begin

if c2 = c2max - 1 {\*实常量表溢出\*}

then fatal(3)

else begin

rconst[c2+1] := x;

c1 := 1; {\*循环变量\*}

while rconst[c1] <> x do

c1 := c1 + 1; {\*查找待登录的实数是否位于表内\*}

if c1 > c2

then c2 := c1 {\*新实数项登录至表\*}

end

end { enterreal };

procedure emit( fct: integer );

{\*这几个emit是用来生成pcode的,fct为操作码,这里为无操作数\*}

begin

if lc = cmax

then fatal(6); {\*超出最大代码长度错误\*}

code[lc].f := fct; {\*with...do...用法 按域赋值\*}

lc := lc + 1

end { emit };

procedure emit1( fct, b: integer );

{\*这几个emit是用来生成pcode的,fct为操作码,这里为1个操作数\*}

begin

if lc = cmax

then fatal(6); {\*超出最大代码长度错误\*}

with code[lc] do {\*按域赋值\*}

begin

f := fct;

y := b;

end;

lc := lc + 1

end { emit1 };

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

{\*这几个emit是用来生成pcode的,fct为操作码,这里为2个操作数\*}

begin

if lc = cmax then fatal(6); {\*超出最大代码长度错误\*}

with code[lc] do {\*按域赋值\*}

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; {\*指令下标\*}

mne: array[0..omax] of

packed array[1..5] of char;

{\*p-code指令\*}

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 ';

{\*以下开始向输出文件制表\*}

writeln(psout);

writeln(psout);

writeln(psout);

{\*输出符号表\*}

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 );

{\*输出分程序表\*}

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 );

{\*输出数组表\*}

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 );

{\*输出p-code序列\*}

writeln( psout, 'code:');

writeln( psout );

for i := 0 to lc-1 do

begin

write( psout, i:5 );

o := code[i]; {\*从code数组中取一个指令\*}

write( psout, mne[o.f]:8, o.f:5 ); {\*按指令的各个域分别输出\*}

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 );

{\*输出起始地址\*}

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

end { printtables };

{\*2K+行block开始\*}

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

{\*处理分析分程序\*}

{\*fsys isfun判断是否函数/过程 level嵌套块层深度\*}

type conrec = record

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，直到取出来的符号等于给定的符号集，打印错误类型\*}

begin

error(n);

skipflag := true; {\*设置允许跳读标志位为真\*}

while not ( sy in fsys ) do

insymbol; {\*当前符号不在允许符号集里就一直跳读\*}

if skipflag then endskip {\*调用在跳读加下划线子程序\*}

end { skip };

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

{\*测试当前符号是否合法,若不合法,打印出错标志并进行跳读\*}

{\*s1为合法符号集 s2为终止符号集\*}

begin

if not( sy in s1 ) {\*测试发现当前符号不在合法符号集中\*}

then skip( s1 + s2, n ) {\*调用跳读程序 传递错误码 扩充允许符号集+终止符号集\*}

end { test };

procedure testsemicolon;

{\*检查分号是否合法\*}

begin

if sy = semicolon {\*合法\*}

then insymbol

else begin

error(14); {\*应是；报错\*}

if sy in [comma, colon] {\*读到了【, :】读下一字符 容错跳读\*}

then insymbol

end;

test( [ident] + blockbegsys, fsys, 6 ) {\*这个测试时什么意思?\*}

{\*分号出口测试 应读到标识符 或者分程序开始符 否则报非法符号错误\*}

end { testsemicolon };

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

{\*登录分程序说明部分出现的名字到tab（与前面enter不同 此处为分程序内使用）\*}

var j,l : integer;

begin

if t = tmax {\*符号表溢出\*}

then fatal(1)

else begin

tab[0].name := id; {\*while循环终止条件（从后向前遍历过程）\*}

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

{\*用分程序索引表display索引当前层分程序 last找到该分程序最后一个标识符在tab中位置\*}

l := j; {\*上一标识符节点为后文生成链表使用\*}

while tab[j].name <> id do

j := tab[j].link; {\*link为指向前面读到表项的指针\*}

if j <> 0 {\*直到在tab中找到id结束\*}

then error(1) {\*如果j不是0 则说明符号表中已经存在该项 重复定义错误\*}

else begin

t := t + 1; {\*登记到表的下一项\*}

with tab[t] do

begin {\*登录表信息\*}

name := id;

link := l; {\*链表前一项\*}

obj := k;

typ := notyp;

ref := 0;

lev := level;

adr := 0;

normal := false { initial value }

end;

btab[display[level]].last := t {\*更新当前分程序层最后节点信息\*}

end

end

end { enter };

function loc( id: alfa ):integer;

{\*查找标识符id在tab中的位置并返回\*}

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

begin

i := level; {\*从当前层开始查找\*}

tab[0].name := id; { sentinel }{\*同样作为while循环终止条件\*}

repeat

j := btab[display[i]].last; {\*在当前层从后望前查找\*}

while tab[j].name <> id do

j := tab[j].link;

i := i - 1;

until ( i < 0 ) or ( j <> 0 ); {\*当层数小于零或者在该层找到了标识符（j!=0)结束\*}

if j = 0

then error(0); {\*遍历完所有层未找到该标识符 未定义错误\*}

loc := j {\*返回位置j ??层数信息？？应该是直接保存在link中了\*}

end { loc } ;

procedure entervariable;

{\*将变量名登录到符号表\*}

begin

if sy = ident {\*是标识符\*}

then begin

enter( id, vvariable ); {\*登录为变量类型\*}

insymbol

end

else error(2) {\*非标识符错误\*}

end { entervariable };

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

{\*处理程序中出现的常量,并由参数(c)返回该常量的类型和数值\*}

var x, sign : integer;

begin

c.tp := notyp; {\*初始化返回参数\*}

c.i := 0;

test( constbegsys, fsys, 50 ); {\*检查当前符号sy是否合法并跳读\*}

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 {\*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

end

else begin

sign := 1;{\*符号设置为1正 -1负\*}

if sy in [plus, minus] {\*是+-号\*}

then begin

if sy = minus

then sign := -1;{\*-\*}

insymbol

end;

if sy = ident {\*是标识符\*}

then begin

x := loc(id); {\*找到常量标识符在tab中的位置\*}

if x <> 0 {\*在tab中找到 （loc查不到返回0）\*}

then

if tab[x].obj <> konstant {\*非常数类型错误\*}

then error(25) {\*常量定义必须是常数或常量标识符\*}

else begin

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

{\*填入常量的类型\*}

if c.tp = reals

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

{\*对于实常量adr存放它在实常量表中登录位置\*}

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

end;

insymbol

end

else if sy = intcon {\*整常量\*}

then begin

c.tp := ints;

c.i := sign\*inum;

insymbol

end

else if sy = realcon {\*实常量\*}

then begin

c.tp := reals;

c.r := sign\*rnum;

insymbol

end

else skip(fsys,50) {\*不是常量错误 跳读\*}

end;

test(fsys,[],6) {\*出口处进行测试常量定义后出现的非法符号并跳读\*}

end

end { constant };

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

{\*处理类型描述,由参数传回它的类型(tp)指向类型详细信息表的指针(rf)和该类型大小(sz)\*}

var eltp : types; {\*元素类型\*}

elrf, x : integer;

elsz, offset, t0, t1 : integer;

procedure arraytyp( var aref, arsz: integer );

{\*登录数组类型到atab并返回数组的登录位置aref和数组大小arsz\*}

var eltp : types; {\*数组内的数据类型\*}

low, high : conrec; {\*数组的上下界(参数类型）\*}

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

begin

constant( [colon, rbrack, rparent, ofsy] + fsys, low );{\*读下界\*}

{\*处理程序中出现的常量,并由参数(c)返回该常量的类型和数值\*}

if low.tp = reals {\*实数上下界非法\*}

then begin

error(27);

low.tp := ints; {\*修正成整型\*}

low.i := 0

end;

if sy = colon {\*应读到..记成colon型\*}

then insymbol

else error(13);

constant( [rbrack, comma, rparent, ofsy ] + fsys, high ); {\*读数组上界\*}

if high.tp <> low.tp {\*数组上下界类型不一致错误\*}

then begin

error(27);

high.i := low.i {\*类型修正\*}

end;

enterarray( low.tp, low.i, high.i ); {\*登录atab数组信息表\*}

aref := a; {\*获得该数组在atab中的索引值\*}

if sy = comma {\*后面接逗号说明是多维数组\*}

then begin

insymbol;

eltp := arrays; {\*数组中每个元素类型都是数组\*}

arraytyp( elrf, elsz ) {\*递归调用\*}

end

else begin

if sy = rbrack {\*右中括号处理\*}

then insymbol

else begin

error(12); {\*未读到数组结尾】错误\*}

if sy = rparent {\*若是）则读下一符号\*}

then insymbol

end;

if sy = ofsy {\*读到of\*}

then insymbol

else error(8); {\*缺少of关键字\*}

typ( fsys, eltp, elrf, elsz )

{\*处理当前的数组元素类型,在typ里面更改eltp、elrf、elsz\*}

end;

with atab[aref] do {\*按索引值将记录数组相关信息填写到atab表\*}

begin

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

size := arsz;

eltyp := eltp;

elref := elrf;

elsize := elsz

end

end { arraytyp };

begin { typ } {\*类型处理过程开始\*}

tp := notyp; {\*用以存储变量的类型\*} {\*初始化\*}

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 ); {\*类型入口测试 开头必须是标识符 array record\*}

if sy in typebegsys

then begin

if sy = ident {\*标识符\*}

then begin

x := loc(id); {\*查表找位置\*}

if x <> 0 {\*查找成功\*}

then with tab[x] do

if obj <> typel {\*该标识符不是类型标识符\*}

then error(29)

else begin {\*正确情况 记录相应信息\*}

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存储空间\*}

if tp = notyp {\*未定义类型错误\*}

then error(30)

end;

insymbol

end

else if sy = arraysy {\*数组类型\*}

then begin

insymbol;

if sy = lbrack {\*读【 再读一个字符\*}

then insymbol

else begin

error(11); {\*缺【\*}

if sy = lparent {\*如果是误打成( 跳读\*}

then insymbol

end;

tp := arrays; {\*记录成数组型\*}

arraytyp(rf,sz) {\*登录数组类型\*}

end

else begin {\*records 类型\*}

insymbol;

enterblock;{\*登录分程序表\*}

tp := records; {\*当前类型设置为records类型\*}

rf := b; {\*rf指向block表中的位置\*}

if level = lmax {\*层数达到最大 表溢出\*}

then fatal(5);

level := level + 1; {\*层数深入 +1\*}

display[level] := b;{\*修改display分程序索引 该层索引到b\*}

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 {\*标识符\*}

then begin

t0 := t; {\*获取当前tab指针的位置\*}

entervariable; {\*信息登录到变量表上 其内部enter后面有一个insymbol\*}

while sy = comma do

begin {\*直到不是，\*}

insymbol; {\*向后继续读符号一个\*}

entervariable {\*信息登录到变量表上\*}

end;

if sy = colon {\*变量说明应以：分割\*}

then insymbol

else error(5); {\*不是：错误\*}

t1 := t; {\*获取当前tab的最新索引\*}

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++\*}

with tab[t0] do {\*修改当前表项\*}

begin

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

ref := elrf; {\*给ref赋值\*}

normal := true;

{\*给normal标记赋值,所有normal的只有变量形参才填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 {\*读到end结束\*}

then begin

if sy = semicolon {\*处理到行尾; 此时还没读到end\*}

then insymbol

else begin

error(14); {\*不是；错误 若读到，进行容错\*}

if sy = comma

then insymbol

end;

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

{\*开启下一行处理时检验当前符号是否合法\*}

end

end; { field section }{\*record while循环处理结束\*}

btab[rf].vsize := offset;

{\*vsize 为运行栈中变量所占存储单元数\*}

{\*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;

{\*psize是参数在运行栈中所占存储单元数\*}

{\*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; {\*读下一个sym\*}

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

end; { record }

test( fsys, [],6 )

{\*类型定义出口测试 检查当前sym是否合法\*}

end;

end { typ };

procedure parameterlist; { formal parameter list }

{\*处理过程或者函数说明中的形参表,将形参及其有关信息登录到符号表中\*}

var tp : types;

valpar : boolean;

rf, sz, x, t0 : integer;

begin

insymbol;

tp := notyp;

rf := 0; {\*初始化符号表的位置\*}

sz := 0; {\*初始化元素的大小\*}

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

{\*合法符号 标识符 var； 终止符号右括号\*}

while sy in [ident, varsy] do

begin

if sy <> varsy

then valpar := true {\*如果前面没有var定义 则是一个值形参\*}

else begin {\*前有var 是一个变量形参\*}

insymbol; {\*读var后面的一个符号\*}

valpar := false

end;

t0 := t; {\*tab表索引\*}

entervariable; {\*参数变量登录到变量表\*}

while sy = comma do {\*参数用，进行分割\*}

begin

insymbol;

entervariable; {\*参数变量登录到变量表\*}

end;

if sy = colon {\*：\*}

then begin

insymbol;

if sy <> ident {\*冒号后面应跟一个类型标识符\*}

then error(2)

else begin

x := loc(id); {\*查这个标识符在表中的位置\*}

insymbol;

if x <> 0

then with tab[x] do

if obj <> typel {\*不是类型标识符错误\*}

then error(29)

else begin

tp := typ; {\*获取参数的类型\*}

rf := ref; {\*获取参数在当前符号表的位置\*}

if valpar {\*如果是值形参\*}

then sz := adr {\*sz获得当前形参在符号表中的位置\*}

else sz := 1 {\*是变量形参 则将sz置为1\*}

{\*在这篇代码中,变量形参的大小都设置为了1\*}

end;

end;

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

{\*参数读取出口测试 合法符号右括号 分号 终止符号 ， 标识符\*}

end

else error(5);{\*类型说明缺少：\*}

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 只有变量形参为false\*}

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

else begin

error(14); {\*非分号错误\*}

if sy = comma {\*如果是逗号,做跳读容错处理\*}

then insymbol

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 均不是则报非法符号 并跳读\*}

end

end { while };

if sy = rparent {\*参数声明结束后应当用右括号结尾\*}

then begin

insymbol;

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

{\*参数读入出口测试 用分号结束或使用冒号声明返回值类型 否则报非法符号错误\*}

end

else error(4) {\*不是右括号结尾,报错\*}

end { parameterlist };

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

var c : conrec;

begin

insymbol;

test([ident], blockbegsys, 2 );

{\*常量定义入口测试 以标识符开始\*}

while sy = ident do

begin

enter(id, konstant); {\*将id登录至符号表,类型为konstant 常量\*}

insymbol;

if sy = eql {\*常量用=定义\*}

then insymbol

else begin

error(16); {\*=和:=的混用错误\*}

if sy = becomes {\*：=的容错处理\*}

then insymbol

end;

constant([semicolon,comma,ident]+fsys,c);

{\*处理程序中出现的常量,并由参数(c)返回该常量的类型和数值 以下根据c填写符号表\*}

tab[t].typ := c.tp;{\*填入类型\*}

tab[t].ref := 0; {\*常量ref为0\*}

if c.tp = reals {\*实型的填表\*}

then begin

enterreal(c.r);

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

end

else tab[t].adr := c.i; {\*整型填表\*}

testsemicolon {\*常量定义结束 检查分号\*}

end

end { constdec };

procedure typedeclaration;

{\*处理类型定义,并将类型名及相应信息填入符号表\*}

var tp: types;

rf, sz, t1 : integer;

begin

insymbol;

test([ident], blockbegsys,2 ); {\*类型定义入口测试 以标识符开始\*}

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

begin

enter(id, typel); {\*标识符的名称的type类型登录tab表\*}

t1 := t; {\*t1读tab表索引\*}

insymbol;

if sy = eql {\*获取等号 继续向下读取\*}

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\*}

{\*typ处理类型描述,由参数传回它的类型(tp),

指向类型详细信息表的指针(rf)和该类型大小(sz)\*}

with tab[t1] do {\*将返回值填表\*}

begin

typ := tp;

ref := rf;

adr := sz

end;

testsemicolon {\*type定义以;结尾\*}

end

end { typedeclaration };

procedure variabledeclaration;

{\*处理变量声明 并将变量名及其相应信息填入符号表\*}

{\*以下操作与前若干子程序相似\*}

var tp : types;

t0, t1, rf, sz : integer;

begin

insymbol;

while sy = ident do

begin

t0 := t; {\*t0表开始位置\*}

entervariable;

while sy = comma do {\*读到，循环读标识符\*}

begin

insymbol;

entervariable;

end;

if sy = colon {\*读到： 之后需要说明类型\*}

then insymbol

else error(5); {\*类型说明缺少：错误\*}

t1 := t; {\*填完一行变量的符号表尾\*}

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\*}

{\*typ处理类型描述,由参数传回它的类型(tp),

指向类型详细信息表的指针(rf)和该类型大小(sz)\*}

while t0 < t1 do {\*对一行变量循环填表\*}

begin

t0 := t0 + 1;

with tab[t0] do

begin

typ := tp;

ref := rf;

lev := level;

adr := dx;

normal := true; {\*都是变量ture\*}

dx := dx + sz {\*更新地址\*}

end

end;

testsemicolon {\*变量声明以分号结尾\*}

end { while }

end { variabledeclaration };

procedure procdeclaration;

{\*处理过程或函数声明,将过程(函数)名填入符号表,递归调用block分析处理程序\*}

var isfun : boolean;{\*需要区分处理的是霍城还是函数\*}

begin

isfun := sy = funcsy;{\*是function就给isfun为true\*}

insymbol;

if sy <> ident {\*过程/函数应以标识符开始\*}

then begin

error(2);

id :=' ' {\*标识符10长度清空\*}

end;

if isfun {\*将标识符及其类型填入符号表\*}

then enter(id,funktion) {\*登录为函数类型\*}

else enter(id,prozedure); {\*登录为过程类型\*}

tab[t].normal := true;

insymbol;

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

{\*直接调用block处理过程\*}

if sy = semicolon {\*读到分号一个procedure/function结束\*}

then insymbol

else error(14);

emit(32+ord(isfun)) {exit} {\*生成p-code 操作码 32+ord(isfun)\*}

end { proceduredeclaration };

procedure statement( fsys:symset );

{\*分析处理各种语句\*}

var i : integer;

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

{\*分析处理表达式,由参数(x)返回求值结果的类型\*}

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

{\*处理结构变量 v为待处理的结构体\*}

var x : item;

a,j : integer;

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

{\*当前的符号应该是( , [ , .\*}

repeat

if sy = period {\*.\*} {\*域索引 用.\*}

{\*(因为引用成员变量的方式为'记录名.成员名'),

因此识别到'.'之后应该开始处理后面的结构名称\*}

then begin

insymbol; { field selector }

if sy <> ident {\*后面不是标识符\*}

then error(2)

else begin

if v.typ <> records {\*找不到这样的记录\*}

then error(31)

else begin { search field identifier }

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; {\*从后向前查符号表tab\*}

while tab[j].name <> id do

j := tab[j].link;

if j = 0

then error(0); {\*未找到 未定义错误\*}

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

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

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

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

then emit1(9,a)

end;

insymbol

end

end

else begin { array selector }

{\*处理数组下标\*}

if sy <> lbrack {\*数组下标必须以【开始\*}

then error(11);

repeat {\*多维数组处理\*}

insymbol;

expression( fsys+[comma,rbrack],x);

{\*处理表达式 array[1+k]\*}

if v.typ <> arrays {\*v类型错误 不是数组\*}

then error(28)

else begin

a := v.ref; {\*从v中获得该数组在atab中的位置\*}

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

{\*如果传入的下标类型和atab中记录的数组下标类型不符则报错\*}

then error(26)

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

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

else emit1(21,a); {\*21 IXX 取下标变量地址\*}

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

v.ref := atab[a].elref {\*填写v各个域\*}

end

until sy <> comma; {\*直到没有遇到逗号终止循环 有，表示多维数组\*}

if sy = rbrack {\*数组结尾】\*}

then insymbol

else begin

error(12);

if sy = rparent {\*对）容错处理\*}

then insymbol

end

end

until not( sy in[lbrack, lparent, period]); {\*repeat 读结构体/数组索引符\*}

test( fsys,[],6) {\*出口测试 检查后续是否有非法字符\*}

end { selector };

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

{\*处理非标准的过程或者函数调用 其中i表示需要调用的过程或者函数名在符号表中的位置\*}

var x : item; {\*代表传进来的参数结构体\*}

lastp,cp,k : integer;

begin

emit1(18,i); { mark stack } {\*生成一条标记栈指令\*}

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

{\*获取该过程或函数最后一个参数在tab表中的位置 用来后面判断参数是否处理完\*}

cp := i; {\*记录当前被调用函数或者过程在符号表中的位置\*}

{\*cp 到 lastp 之间是tab表中形参列表,

若是在循环中cp > latp则表示实参与形参个数不一致\*}

if sy = lparent {\*（\*}

then begin { actual parameter list }

{\*如果识别到左括号则开始进行被调用过程的参数处理 实参处理\*}

repeat

insymbol;

if cp >= lastp {\*形参与实参个数不匹配\*}

then error(39)

else begin

cp := cp + 1;

if tab[cp].normal

{\*如果normal值为真 则表示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);

{\*调用表达式处理子过程\*}

if x.typ = tab[cp].typ

{\*实际传参类型与记录的参数类型如果相同\*}

then begin

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

{\*表达式指向的btab表或atab表中位置与符号表记录中的位置不同,应该是数组或者记录的形参与实参不一致？\*}

then error(36)

else if x.typ = arrays

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

{\*生成一条数组记录指令 装入块\*}

else if x.typ = records

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

{\*生成一条record指令 装入块\*}

end

else if ( x.typ = ints ) and ( tab[cp].typ = reals ) {\*如果传入的是整型,而形参为实数型\*}

then emit1(26,0) {\*实数转浮点数\*}

else if x.typ <> notyp {\*实参和形参对应类型相同\*}

then error(36);

end

else begin { variable parameter }

{\*normal为false 传入的是var定义的变量形参 以下处理变量形参\*}

if sy <> ident {\*如果传进来的不是标识符则报错\*}

then error(2)

else begin

k := loc(id); {\*找到其在tab中的位置\*}

insymbol;

if k <> 0

then begin

if tab[k].obj <> vvariable

{\*如果传进去的参数不是变量类型则报错\*}

then error(37);

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

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

{\*给传进来的record赋予查表信息\*}

if tab[k].normal {\*传进来的指针指的是否是一个指针\*}

then emit2(0,tab[k].lev,tab[k].adr) {\*传进来的指针指的是一个指针\*}

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

if sy in [lbrack, lparent, period]

{\*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);

{\*处理数组或结构\*}

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

{\*实参和形参对应类型不相同\*}

then error(36)

end

end

end {variable parameter }

end;

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

until sy <> comma; {\*循环直到处理完逗号\*}

if sy = rparent {\*参数表一右括号结尾\*}

then insymbol

else error(4)

end;

if cp < lastp {\*实参小于形参数\*}

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

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

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

then emit2(3,tab[i].lev, level ) {\*生成3号指令更新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(a + b = c, 根据a 和 b 判断c的类型)\*}

begin

if ( a > reals ) or ( b > reals )

{\*该算数表达式的类型不合法\*}

then begin

error(33);

resulttype := notyp

end

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

{\*有一个为无类型\*}

then resulttype := notyp {\*则返回无类型\*}

else if a = ints

then if b = ints

then resulttype := ints

else begin

resulttype := reals;

emit1(26,1) {\*生成指令 需要转换类型为浮点数\*}

end

else begin {\*只要a,b中有一个为real 则应返回real\*}

resulttype := reals;

if b = ints

then emit1(26,0) {\*转化为浮点型\*}

end

end { resulttype } ;

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

{\*处理表达式,由x返回求值结果的类型\*}

var y : item;

op : symbol;

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

{\*处理简单表达式(+、-、or),由参数x返回求值结果的类型\*}

var y : item;

op : symbol;

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

{\*处理项,由参数x返回结果类型\*}

var y : item;

op : symbol;

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

{\*处理因子,由参数x返回结果类型\*}

var i,f : integer;

procedure standfct( n: integer );

{\*处理标准函数调用,传入标准函数的编号n,执行不同的操作\*}

var ts : typset; {\*类型集合 type set\*}

begin { standard function no. n }

if sy = lparent {\*标准函数调用以左括号开始\*}

then insymbol

else error(9);

if n < 17

then begin

expression( fsys+[rparent], x ); {\*递归的参数处理\*}

case n of {\*以下对标准函数按编号处理\*}

{ abs, sqr } 0,2: begin

ts := [ints, reals];

{\*绝对值 平方类型整型或实型\*}

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

{\*登录类型\*}

if x.typ = reals {\*是实型n++？作用？\*}

then n := n + 1

end;

{ odd, chr } 4,5: ts := [ints];

{ odr } 6: ts := [ints,bools,chars];

{ succ,pred } 7,8 : begin

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

then emit1(26,0) {\*如果是整数则转换为浮点数\*}

end;

{\*以上case根据不同的standfct决定类型集合\*}

end; { case }

if x.typ in ts {\*类型是在类型集中的\*}

then emit1(8,n) {\*生成一条标准函数指令\*}

else if x.typ <> notyp

then error(48); {\*函数表达式的变元表示不正确\*}

end

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

if sy <> ident

then error(2)

else if id <> 'input ' {\*在main中定义\*}

then error(0) {\*应是文件指针类型的标识符 否则标识符未定义\*}

else insymbol;

emit1(8,n);

end;

x.typ := tab[i].typ; {\*返回符号表中记录的类型\*}

if sy = rparent {\*标准函数处理以）结尾\*}

then insymbol

else error(4)

end { standfct } ;

{\*以下开始因子处理\*}

begin { factor }

x.typ := notyp;

x.ref := 0;

test( facbegsys, fsys, 58 );

{\*因子入口测试 检查首符号是否合法\*}

while sy in facbegsys do

begin

if sy = ident {\*标识符\*}

then begin

i := loc(id); {\*查表\*}

insymbol;

with tab[i] do

case obj of {\*因子是常数\*}

konstant: begin

x.typ := typ;

x.ref := 0;

if x.typ = reals

then emit1(25,adr) {\*装入实数\*}

else emit1(24,adr) {\*装入字面常量\*}

end;

vvariable: begin {\*因子是变量\*}

x.typ := typ;

x.ref := ref;

if sy in [lbrack, lparent,period]

{\*如果该标识符变量后面有'('、']'、'.'

则说明该变量存在子结构\*}

then begin

if normal

then f := 0 {\*非变量形参为加载地址\*}

else f := 1; {\*变量形参加载值\*}

emit2(f,lev,adr); {\*生成0 或 1指令\*}

selector(fsys,x);

{\*处理子结构 返回到x\*}

if x.typ in stantyps

then emit(34) {\*标准类型取栈顶值\*}

end

else begin {\*无子结构\*}

if x.typ in stantyps

then if normal

then f := 1 {\*标准类型的值形参进行存值操作\*}

else f := 2 {\*标准类型的变量形参进行间接存值操作\*}

else if normal

then f := 0 {\*非标准类型的值形参取地址\*}

else f := 1; {\*非标准类型的变量形参取值\*}

emit2(f,lev,adr) {\*根据上述判断的f生成指令\*}

end

end;

typel,prozedure: error(44); {\*因子标识符不能是type或者过程\*}

funktion: begin

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 ]

{\*因子常量类型\*}

then begin

if sy = realcon

then begin {\*实常量处理常数\*}

x.typ := reals;

enterreal(rnum); {\*填实数表\*}

emit1(25,c1) {\*装入实数\*}

end

else begin

if sy = charcon {\*字符常量\*}

then x.typ := chars

else x.typ := ints; {\*整常量\*}

emit1(24,inum) {\*装入字面常量\*}

end;

x.ref := 0;

insymbol

end

else if sy = lparent {\*（\*}

then begin

insymbol;

expression(fsys + [rparent],x);

{\*读到左括号有嵌套结构 递归调用查因子\*}

if sy = rparent {\*右括号结尾匹配\*}

then insymbol

else error(4)

end

else if sy = notsy {\*读到一个not\*}

then begin

insymbol;

factor(fsys,x); {\*递归因子处理\*}

if x.typ = bools {\*返回结果是bool\*}

then emit(35) {\*生成逻辑非指令\*}

else if x.typ <> notyp

then error(32) {\*not,and或or的操作数必须是boolean型\*}

end;

test(fsys,facbegsys,6)

{\*因子出口测试 非法字符\*}

end { while }

end { factor };

{\*以下开始处理因子\*}

begin { term }

factor( fsys + [times,rdiv,idiv,imod,andsy],x);

{\*在因子的合法集上扩充\*}{\*x为运算符1\*}

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

{\*如果因子后面跟符号'\*''/''div''mod''and',说明后面还有因子,进入循环\*}

begin

op := sy; {\*记当前符号为操作符\*}

insymbol;

factor(fsys+[times,rdiv,idiv,imod,andsy],y );

{\*递归调用因子处理\*}{\*y为运算符2\*}

if op = times {\*‘\*’\*}

then begin

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

case x.typ of

notyp: ; {\*无类型不操作\*}

ints : emit(57); {\*乘整数指令\*}

reals: emit(60); {\*乘实数指令\*}

end

end

else if op = rdiv {\*实型除\*}

then begin

if x.typ = ints {\*整型 需要转实型\*}

then begin

emit1(26,1); {\*转浮点指令\*}

x.typ := reals;

end;

if y.typ = ints

then begin

emit1(26,0); {\*y转浮点\*}

y.typ := reals;

end;

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

then emit(61) {\*都是实型直接生成除指令\*}

else begin

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

then error(33); {\*类型不合法 不是实型 不为无类型\*}

x.typ := notyp {\*修改第一个操作数类型\*}

end

end

else if op = andsy {\*&\*}

then begin

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

then emit(56) {\*生成与操作 必须为bool型\*}

else begin

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

then error(32); {\*&|!必须bool 类型非法\*}

x.typ := notyp

end

end

else begin { op in [idiv,imod] } {\*整型除 取模\*}

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

then if op = idiv {\*/\*}

then emit(58) {\*整型除 取模\*}

else emit(59)

else begin

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

then error(34); {\*非法类型\*}

x.typ := notyp

end

end

end { while }

end { term };

begin { simpleexpression }

{\*处理简单表达式\*}

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

then begin

op := sy;

insymbol;

term( fsys+[plus,minus],x); {\*处理项\*}

if x.typ > reals

{\*类型是bools,chars,arrays, records\*}

then error(33) {\*不是算数运算类型 报错\*}

else if op = minus

then emit(36) {\*求负\*}

end

else term(fsys+[plus,minus,orsy],x); {\*调用项处理\*}

while sy in [plus,minus,orsy] do {\*后面紧跟+-| 后面还有项\*}

begin

op := sy;

insymbol;

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

if op = orsy

then begin

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

then emit(51) {\*生成or操作指令 必须全bool\*}

else begin

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

then error(32); {\*类型\*}

x.typ := notyp

end

end

else begin

x.typ := resulttype(x.typ,y.typ); {\*+-需要处理返回值类型\*}

case x.typ of

notyp: ;

ints: if op = plus

then emit(52) {\*生成整型+-指令\*}

else emit(53);

reals:if op = plus {\*生成实型+-指令\*}

then emit(54)

else emit(55)

end { case }

end

end { while }

end { simpleexpression };

begin { expression }{\*复杂表达式处理\*}

simpleexpression(fsys+[eql,neq,lss,leq,gtr,geq],x); {\*调用简单表达是处理\*}

if sy in [ eql,neq,lss,leq,gtr,geq]

then begin

op := sy;

insymbol;

simpleexpression(fsys,y); {\*调用简单表达是处理\*}

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

{\*类型相同 且整型、布尔型和字符型都可以借用整型的运算\*}

then case op of

eql: emit(45);

neq: emit(46);

lss: emit(47);

leq: emit(48);

gtr: emit(49);

geq: emit(50); {\*根据操作符生成相应指令\*}

end

else begin {\*类型不同\*}

if x.typ = ints

then begin

x.typ := reals;

emit1(26,1) {\*转浮点\*}

end

else if y.typ = ints

then begin

y.typ := reals;

emit1(26,0) {\*类型转换\*}

end;

if ( x.typ = reals)and(y.typ=reals) {\*如果都成功的转成实型\*}

then case op of

eql: emit(39);

neq: emit(40);

lss: emit(41);

leq: emit(42);

gtr: emit(43);

geq: emit(44); {\*则可生成相应指令\*}

end

else error(35)

end;

end

end { expression };

procedure assignment( lv, ad: integer );

{\*处理赋值语句 参数lv层深 ad偏移值\*}

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); {\*变量形参存值 值形参存地址\*}

if sy in [lbrack,lparent,period]

then selector([becomes,eql]+fsys,x); {\*子结构分析\*}

if sy = becomes {\*：=\*}

then insymbol

else begin

error(51);

if sy = eql {\*=容错\*}

then insymbol

end;

expression(fsys,y); {\*读表达式分析\*}

if x.typ = y.typ

then if x.typ in stantyps {\*标准类型 xy一致\*}

then emit(38) {\*存值\*}

else if x.ref <> y.ref

then error(46) {\*被赋值变量应该与表达式类型相同\*}

else if x.typ = arrays {\*数组或是record\*}

then emit1(23,atab[x.ref].size) {\*复制块\*}

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

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

then begin

emit1(26,0); {\*统一成浮点数\*}

emit(38) {\*存值\*}

end

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

then error(46) {\*类型错误\*}

end { assignment };

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

begin

insymbol;

statement([semicolon,endsy]+fsys);

while sy in [semicolon]+statbegsys do {\*下一符号是表达式开头 循环读表达式\*}

begin

if sy = semicolon {\*分号往下读\*}

then insymbol

else error(14);

statement([semicolon,endsy]+fsys) {\*表达式\*}

end;

if sy = endsy {\*end\*}

then insymbol

else error(57)

end { compoundstatement };

procedure ifstatement;

{\*处理if\*}

var x : item;

lc1,lc2: integer;

begin

insymbol;

expression( fsys+[thensy,dosy],x); {\*处理exp\*}

if not ( x.typ in [bools,notyp] ) {\*该表达式不是bool 错误\*}

then error(17);

lc1 := lc; {\*记录当前的pc值 便于跳转使用\*}

emit(11); { jmpc } {\*为假跳转\*}

if sy = thensy {\*then\*}

then insymbol

else begin

error(52); {\*少then\*}

if sy = dosy {\*do容错处理\*}

then insymbol

end;

statement( fsys+[elsesy]); {\*读表达式\*}

if sy = elsesy

then begin

insymbol;

lc2 := lc; {\*记pc值 跳转调用\*}

emit(10); {\*无条件跳转\*}

code[lc1].y := lc; {\*更改相应pc值处的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; {\*j用于exittab的索引下标\*}

casetab : array[1..csmax]of {\*最大30个case\*}

packed record

val,lc : index

end; {\*case表\*}

exittab : array[1..csmax] of integer; {\*退出表\*}

procedure caselabel;

{\*处理case语句中的标号 将各标号对应的目标代码地址填入casetab表中

并检查符号有无重复定义\*}

var lab : conrec;

k : integer;

begin

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

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

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

else if i = csmax {\*case表溢出\*}

then fatal(6)

else begin

i := i+1; {\*casestatement的变量i为case总数\*}

k := 0; {\*用来检查是否重复定义的变量\*}

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

casetab[i].lc := lc; {\*保存新case后面代码的位置\*}

repeat

k := k+1

until casetab[k].val = lab.i; {\*在之前的表中找到了相同的值\*}

if k < i {\*重定义错误\*}

then error(1); { multiple definition }

end

end { caselabel };

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

begin

if sy in constbegsys {\*常量\*}

then begin

caselabel; {\*一个标签\*}

while sy = comma do {\*每个，反复读标签\*}{\*1case对应多标签\*}

begin

insymbol;

caselabel

end;

if sy = colon {\*结束的：\*}

then insymbol

else error(5);

statement([semicolon,endsy]+fsys);{\*处理case表达式\*}

j := j+1; {\*j++\*}

exittab[j] := lc;

{\*记录当前case分支结束的代码位置 即跳转指令的位置\*}{\*跳转都需要记录代码位置\*}

emit(10) {\*无条件跳转\*}

end

end { onecase };

begin { casestatement }

{\*语法case ...(第一个case分支,再之后的就没有of了)of ...: begin .....end\*}

insymbol;

i := 0;

j := 0;

expression( fsys + [ofsy,comma,colon],x ); {\*读表达式\*}

if not( x.typ in [ints,bools,chars,notyp ]) {\*类型要求\*}

then error(23);

lc1 := lc; {\*跳转需记录位置\*}

emit(12); {jmpx}

if sy = ofsy {\*第一个case后要有of\*}

then insymbol

else error(8);

onecase; {\*处理一个case\*}

while sy = semicolon do {\*不同case之间用；分割 循环处理一个case\*}

begin

insymbol;

onecase

end;

code[lc1].y := lc; {\*确定了情况跳转到的表的开始地址之后对之前声明的代码进行回填\*}

for k := 1 to i do

begin

emit1( 13,casetab[k].val); {\*循环确定case表的登记项\*}

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

end;

emit1(10,0); {\*跳转\*}

for k := 1 to j do

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

{\*设置每个case分支结束后的跳转地址

此时lc指向情况表结束之后的位置 所有的分支结束后都跳到这里\*}

if sy = endsy {\*case的结束标志\*}

then insymbol

else error(57)

end { casestatement };

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

var x : item;

lc1: integer;

begin

lc1 := lc; {\*涉及跳转记录pc\*}

insymbol;

statement( [semicolon,untilsy]+fsys); {\*读语句\*}

while sy in [semicolon]+statbegsys do {\*repeat中的语句用；分割\*}

begin

if sy = semicolon

then insymbol

else error(14);

statement([semicolon,untilsy]+fsys)

end;

if sy = untilsy {\*until\*}

then begin

insymbol;

expression(fsys,x);

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

{\*处理until后面的表达式 必须是boolean类型\*}

then error(17);

emit1(11,lc1);

{\*跳转指令 如果表达式为假 则跳到repeat开始的位置lc1继续循环执行\*}

end

else error(53) {\*应该是until\*}

end { repeatstatement };

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

var x : item;

lc1,lc2 : integer;

begin

insymbol;

lc1 := lc; {\*跳转while首地址\*}

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

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

then error(17);

lc2 := lc; {\*当前位置\*}

emit(11);

if sy = dosy {\*do\*}

then insymbol

else error(54);

statement(fsys); {\*处理语句\*}

emit1(10,lc1); {\*条件跳转会while 头\*}

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

then begin

i := loc(id); {\*找到其在tab表中的位置\*}

insymbol;

if i = 0

then cvt := ints {\*未找到定义 计数变量类型默认为整型\*}

else if tab[i].obj = vvariable {\*var\*}

then begin

cvt := tab[i].typ; {\*计数变量类型\*}

if not tab[i].normal{\*变量形参\*}

then error(37)

else emit2(0,tab[i].lev, tab[i].adr );{\*取数\*}

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

then error(18) {\*循环变量类型限定\*}

end

else begin

error(37);

cvt := ints {\*统一使用int类型表示\*}

end

end

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

if sy = becomes {\*：=\*}

then begin

insymbol;

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

{\*for i:= 1 + 2 to 3 do ... 此处处理1+2表达式并将结果的类型返回\*}

if x.typ <> cvt {\*与循环变量类型不一致错误\*}

then error(19);

end

else skip([tosy, downtosy,dosy]+fsys,51); {\*跳过中间的to/downto\*}

f := 14; {\*增量型\*}

if sy in [tosy,downtosy]

then begin

if sy = downtosy

then f := 16; {\*改为减量型\*}

insymbol;

expression([dosy]+fsys,x); {\*读表达式\*}

if x.typ <> cvt {\*不同的类型\*}

then error(19)

end

else skip([dosy]+fsys,55); {\*跳读do\*}

lc1 := lc; {\*跳转需要记录代码位置\*}

emit(f);

if sy = dosy

then insymbol

else error(54);

lc2 := lc; {\*记录for出口位置\*}

statement(fsys); {\*处理do内部语句\*}

emit1(f+1,lc2); {\*f+1再入口测试指令\*}

code[lc1].y := lc {\*填code表\*}

{\*若进不了第一次入口测试则直接跳过循环体\*}

{\*其实际的PCODE指令设计为

for第一次判断和之后的判断用两个指令进行,再判断相当于do..while形式\*}

end { forstatement };

procedure standproc( n: integer );{\*处理标准的输入输出的过程调用:读.写\*}

var i,f : integer;

x,y : item;

begin

case n of

1,2 : begin { read }

if not iflag

then begin

error(20); {\*程序头应有input/output\*}

iflag := true

end;

if sy = lparent {\*读从(开始\*}

then begin

repeat

insymbol;

if sy <> ident {\*应先读到标识符\*}

then error(2)

else begin

i := loc(id);

insymbol; {\*查该标识符在tab中的位置\*}

if i <> 0

then if tab[i].obj <> vvariable {\*应是变量\*}

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);

{\*生成一个取参数指令\*}

if sy in [lbrack,lparent,period]

{\*存在子结构进行子结构查询\*}

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);

{\*读出口测试\*}

until sy <> comma;

if sy = rparent

then insymbol

else error(4) {\*读以）结束\*}

end;

if n = 2

then emit(62) {\*读完一行\*}

end;

3,4 : begin { write }

if sy = lparent

then begin

repeat

insymbol;

if sy = stringcon {\*写字符串常量\*}

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);

if sy = colon {\*用：指定域宽\*}

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)) {\*隐含域宽写\*}

end

until sy <> comma; {\*循环写用，分割的每一项\*}

if sy = rparent {\*以右括号结束写\*}

then insymbol

else error(4)

end;

if n = 4

then emit(63) {\*换行写\*}

end; { write }

end { case };

end { standproc } ;

begin { statement }

{\*进行处理时分析第一个单词调用不同的子过程 P330图\*}

if sy in statbegsys+[ident]

then case sy of

ident : begin

i := loc(id); {\*首符号为标识符\*}

insymbol;

if i <> 0

then case tab[i].obj of

konstant,typel : error(45); {\*语句不能以常量或类型开始\*}

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

{\*变量开始为赋值语句\*}

prozedure: if tab[i].lev <> 0 {\*不是第0层\*}

then call(fsys,i) {\*过程调用\*}

else standproc(tab[i].adr); {\*标准过程调用\*}

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

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

else error(45) {\*？不懂\*}

end { case }

end;

{\*其他开始符对应的语句\*}

beginsy : compoundstatement;

ifsy : ifstatement;

casesy : casestatement;

whilesy : whilestatement;

repeatsy: repeatstatement;

forsy : forstatement;

end; { case }

test( fsys, [],14);

end { statement };

begin { block }

{\*block块刚一进入开头不是function就是procdure 所以用isfun传参确定\*}

dx := 5; {\*dx为变量存储的索引 初值为5是为了给分程序留出内务信息区\*}

prt := t; {\*获取当前符号表的位置\*}

if level > lmax {\*表溢出\*}

then fatal(5);

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

enterblock; {\*登录分程序信息\*}

prb := b; {\*获取分程序表的位置\*}

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

tab[prt].typ := notyp;

tab[prt].ref := prb;

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

then parameterlist; {\*处理形式参数表\*}

btab[prb].lastpar := t;

btab[prb].psize := dx;

{\*dx在paramaterlist过程里面也进行了改变 在5的基础上增加了形参的总大小\*}

if isfun {\*如果是函数\*}

then if sy = colon {\*如果是冒号则处理返回值类型部分\*}

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)

{\*函数结果必须是integer,real,boolean或char类型\*}

end

else skip( [semicolon]+fsys,2 )

end

else error(5);

if sy = semicolon {\*声明结束要有分号\*}

then insymbol

else error(14);

repeat {\*循环处理变量和函数/过程声明部分\*}

if sy = constsy

then constdec;

if sy = typesy

then typedeclaration;

if sy = varsy

then variabledeclaration;

btab[prb].vsize := dx;

while sy in [procsy,funcsy] do {\*分程序块的嵌套结构\*}

procdeclaration;

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

until sy in statbegsys;

tab[prt].adr := lc;

insymbol;

statement([semicolon,endsy]+fsys); {\*处理代码块中的各条语句\*}

while sy in [semicolon]+statbegsys do

begin

if sy = semicolon {\*语句间；\*}

then insymbol

else error(14);

statement([semicolon,endsy]+fsys);

end;

if sy = endsy {\*分程序结束\*}

then insymbol

else error(57);

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

end { block };

procedure interpret; {\*P-code解释执行程序\*}

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

pc : integer; { program counter } {\*pc值\*}

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

b : integer; { base index } {\*基地址\*}

h1,h2,h3: integer;

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

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

{\*program state 程序状态：run 运行 chk出错检查\*}

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; {\*运行栈定义\*}

procedure dump;{\*程序执行时打印现场剖析信息(display,t,b及运行栈S的内容)\*}

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); {\*打印display\*}

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); {\*打印运行栈\*}

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

end; {dump }

{\*以下inter0-6为PCODE 0-63条指令所对应的操作\*}

procedure inter0;

begin

case ir.f of

{\*0-2中的x为变量所在分程序层次 y为相对地址\*}

0 : begin { load addrss }

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状态设置为stkchk记录错误类型\*}

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

{\*完成取地址 实际地址 = level起始地址+偏移地址,放到栈顶\*}

end;

1 : begin { load value }

t := t + 1;

if t > stacksize

then ps := stkchk

else s[t] := s[display[ir.x]+ir.y] {\*取值放栈顶\*}

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] {\*间接取值\*}

end;

3 : begin { update display }

h1 := ir.y; {\*被调用过程的层次\*}

h2 := ir.x; {\*被调用的过程\*}

h3 := b; {\*栈基地址\*}

repeat

display[h1] := h3; {\*给display赋值\*}

h1 := h1-1; {\*层次--\*}

h3 := s[h3+2].i

until h1 = h2 {\*层次到当前层次停止赋值\*}

end;

8 : case ir.y of {\*标准函数处理\*}

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);

6 : s[t].i := ord(s[t].c);

7 : s[t].c := succ(s[t].c);

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; {\*eof eoln需要更改运行栈\*}

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 }{\*栈顶元素+y\*}

end { case ir.y }

end; { inter0 }

procedure inter1;

var h3, h4: integer;

begin

case ir.f of

10 : pc := ir.y ; { jump } {\*无条件跳转 pc到y\*}

11 : begin { conditional jump }

if not s[t].b {\*栈顶为假跳转\*}

then pc := ir.y;

t := t - 1 {\*栈顶指针--\*}

end;

12 : begin { switch } {\*转移到y查找情况表\*}

h1 := s[t].i; {\*取栈顶元素\*}

t := t-1; {\*栈顶--\*}

h2 := ir.y; {\*转移到的位置\*}

h3 := 0;

repeat

if code[h2].f <> 13

then begin

h3 := 1;

ps := caschk

end

else if code[h2].y = h1 {\*伪指令 情况表的登记项\*}

then begin

h3 := 1;

pc := code[h2+1].y {\*啥意思\*}

end

else h2 := h2 + 2

until h3 <> 0

end;

14 : begin { for1up } {\*增量for入口\*}

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;

14 : begin { for1up }

{\*增量for(to)的初始判断,

其中 s[t-2].i存储的是计数变量的地址,

s[t-1].i存储的是计数变量值(刚进来时是初值),

s[s[t-2].i].i存储的是计数变量的终值\*}

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

if h1 <= s[t].i

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

else begin

t := t - 3; {\*栈指针上移3个（3条语句）\*}

pc := ir.y {\*跳出循环到y处 这里的y由后面的语句回传得到\*}

end

end;

15 : begin { for2up } {\*增量for再入口判断\*}

h2 := s[t-2].i; {\*在入口可以直接加载计数变量的地址到h2\*}

h1 := s[h2].i+1; {\*计数变量++\*}

if h1 <= s[t].i

then begin

s[h2].i := h1; {\*不断增长s[t-1].i的值\*}

pc := ir.y {\*跳出循环\*}

end

else t := t-3;

end;

17 : begin { for2down } {\*与15类似\*}

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 } {\*与16类似\*}

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 } {\*调用\*}

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;

{\*调用使得层次数++ 新建一个层次 并将该层次基址指向当前层次基址\*}

h4 := s[h1+3].i+h1; {\*动态链DL的值\*}

s[h1+1].i := pc; {\*返回地址\*}

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

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

for h3 := t+1 to h4 do

s[h3].i := 0;

b := h1;

t := h4;

pc := tab[h2].adr;

if stackdump {\*打印程序执行现场信息\*}

then dump

end;

end { case }

end; { inter1 }

procedure inter2;

begin

case ir.f of

20 : begin { index1 } {\*取下标变量的地址 元素长度1\*}

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

h2 := atab[h1].low; {\*数组低地址\*}

h3 := s[t].i; {\*数组下标\*}

if h3 < h2

then ps := inxchk {\*数组下标小于低地址 越界\*}

else if h3 > atab[h1].high

then ps := inxchk {\*数组下标大于高地址 越界\*}

else begin

t := t-1; {\*弹运行栈\*}

s[t].i := s[t].i+(h3-h2) {\*基地址+偏移\*}

end

end;

21 : begin { index } {\*取下标变量的地址\*}

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

h2 := atab[h1].low;

h3 := s[t].i;

if h3 < h2

then ps := inxchk

else if h3 > atab[h1].high

then ps := inxchk

else begin

t := t-1;

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

{\*与20区别在于多乘一个元素大小\*}

end

end;

22 : begin { load block } {\*装入块\*}

h1 := s[t].i; {\*取栈顶元素\*}

t := t-1;

h2 := ir.y+t; {\*获取块的分配空间 其中块大小为ir.y\*}

if h2 > stacksize

then ps := stkchk {\*运行栈溢出\*}

else while t < h2 do

begin

t := t+1;

s[t] := s[h1];

h1 := h1+1 {\*循环往块中载入元素\*}

end

end;

23 : begin { copy block } {\*复制块\*}

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

h2 := s[t].i;

h3 := h1+ir.y; {\*ir.y复制块的大小 分配空间\*}

while h1 < h3 do

begin

s[h1] := s[h2];

h1 := h1+1;

h2 := h2+1 {\*寻款拷贝块\*}

end;

t := t-2 {\*弹2位\*}

end;

24 : begin { literal } {\*装入字面常量\*}

t := t+1;

if t > stacksize

then ps := stkchk

else s[t].i := ir.y {\*给栈顶赋值\*}

end;

25 : begin { load real } {\*装入实常量\*}

t := t+1;

if t > stacksize

then ps := stkchk

else s[t].r := rconst[ir.y] {\*读rconst给栈顶赋值\*}

end;

26 : begin { float } {\*转浮点数\*}

h1 := t-ir.y; {\*找到变量位置\*}

s[h1].r := s[h1].i {\*i to r\*}

end;

27 : begin { read } {\*读字符 1int 2real 3char\*}

if eof(prd)

then ps := redchk {\*读到文件尾\*}

else case ir.y of

1 : read(prd, s[s[t].i].i); {\*调用读文件stdfunc\*}

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

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

end;

t := t-1

end;

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

h1 := s[t].i;

h2 := ir.y;

t := t-1;

chrcnt := chrcnt+h1;

if chrcnt > lineleng

then ps := lngchk; {\*长度溢出错误\*}

repeat

write(prr,stab[h2]);

h1 := h1-1;

h2 := h2+1

until h1 = 0

end;

29 : begin { write1 } {\*写字符串 隐含域宽\*}

chrcnt := chrcnt + fld[ir.y]; {\*+域宽\*}

if chrcnt > lineleng

then ps := lngchk {\*输出行溢出\*}

else case ir.y of {\*根据选定域的不同写字符串\*}

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; {\*fin表示停止\*}

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

t := b-1; {\*表最后一项 栈基地址-1\*}

pc := s[b+1].i; {\*pc记成上一层次基址出的值\*}

b := s[b+3].i

end;

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

t := b; {\*表最后项即是 运行栈基址\*}

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

b := s[b+3].i

end;

34 : s[t] := s[s[t].i]; {\*取栈顶元素为地址的内容\*}

35 : s[t].b := not s[t].b; {\*逻辑非\*}

36 : s[t].i := -s[t].i; {\*求负\*}

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); {\*写给定域宽实数\*}

t := t-3 {\*处理了栈上面3个单元向下移动3\*}

end;

38 : begin { store }

s[s[t-1].i] := s[t]; {\*栈顶元素存放到s[t-1]中元素为地址的单元\*}

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; {\*各种运算\*}

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; {\*各种运算\*}

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 {\*除零检查\*}

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

end;

59 : begin

t := t-1;

if s[t+1].i = 0

then ps := divchk {\*除零检查\*}

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) {\*读完一行\*}

then ps := redchk

else readln;

63 : begin {\*换行写\*}

writeln(prr);

lncnt := lncnt+1;

chrcnt := 0;

if lncnt > linelimit

then ps := linchk

end

end { case };

end; { inter6 }

begin { interpret }

s[1].i := 0; {\*解释程序一开始对运行栈进行的初始化\*}

s[2].i := 0;

s[3].i := -1;

s[4].i := btab[1].last; {\*28\*} {\*s[4].i?\*}

display[0] := 0;

display[1] := 0; {\*初始化display表\*}

t := btab[2].vsize-1; {\*栈顶指针\*}

b := 0; {\*基地址\*}

pc := tab[s[4].i].adr; {\*pc\*}

lncnt := 0;

ocnt := 0;

chrcnt := 0; {\*初始化计数变量\*}

ps := run; {\*program state 先置成运行run状态\*}

fld[1] := 10;

fld[2] := 22;

fld[3] := 10;

fld[4] := 1;

repeat

ir := code[pc]; {\*ir为从code集中取指令 code类似于im\*}

pc := pc+1; {\*pc++\*}

ocnt := ocnt+1; {\*指令总数++\*}

case ir.f div 10 of {\*ir.f为指令操作码这个域\*}

0 : inter0;

1 : inter1; {\*除以10寻找处理子程序\*}

2 : inter2;

3 : inter3;

4 : inter4;

5 : inter5;

6 : inter6;

end; { case }

until ps <> run; {\*ps是run一直运行\*}

if ps <> fin {\*错误信息及出错时各种变量信息输出到文件\*}

then begin

writeln(prr); {\*写运行错误讯息\*}

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; {\*按错误类型输出\*}

h1 := b; {\*嵌套层数 基地址\*}

blkcnt := 10; { post mortem dump } {\*层数\*}

repeat

writeln( prr );

blkcnt := blkcnt-1;

if blkcnt = 0

then h1 := 0; {\*如果blkcnt为0\*}

h2 := s[h1+4].i; {\*获取过程或者函数名在tab表中的位置\*}{\*+4？\*}

if h1 <> 0

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

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

{\*获取分程序中说明的最后一个标识符在tab表中的位置\*}

while h2 <> 0 do {\*循环处理分程序中的标识符\*}

with tab[h2] do

begin

if obj = vvariable

then if typ in stantyps {\*标准类型的标识符\*}

then begin

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

if normal

then h3 := h1+adr {\*非变量形参 层数基地址 + 偏移\*}

else h3 := s[h1+adr].i; {\*变量形参 读运行栈信息\*}

case typ of

ints : writeln(prr,s[h3].i); {\*输出错误信息\*}

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

bools: if s[h3].b

then writeln(prr,'true')

else writeln(prr,'false');

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

end

end;

h2 := link

end;

h1 := s[h1+3].i {\*指向分程序的DL,即上一层\*}

until h1 < 0

end;

writeln(prr);

writeln(prr,ocnt,' steps');

end; { interpret }

procedure setup;

{\*建立保留字表和特定字符表\*}

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;

{\*在符号表中登记标准类型、函数和过程的名字以及它们的相应信息\*}

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 }

{\*主程序 编译初始化\*}

setup;{\*建立保留字表和特定字符表\*}

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

{\*常量首符号集：+ - 整常 实常 字符常 标识符\*}

typebegsys := [ ident, arraysy, recordsy ];

{\*类型首符号集：标识符 数组符号 记录符号\*}

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

{\*分程序首符号集：常量 类型 变量 过程 函数 begin\*}

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

{\*因子首符号集：常量 标识符 左括号 not\*}

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

{\*语句首符号集：begin if while repeat for case ??过程 函数 变量？\*}

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

{\*标准类型集\*}

{\*以下为位置指针初始化\*}

lc := 0; {\*初始化PC\*}

ll := 0; {\*初始化当前行的长度\*}

cc := 0; {\*初始化当前行位置指针\*}

ch := ' '; {\*初始化当前符号\*}

errpos := 0;{\*初始化错误位置\*}

errs := []; {\*初始化错误集合\*}

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

{\*提示用户输入将要编译的源程序文件和编译输出文件的名字\*}

writeln;

write( 'Source input file ?');

readln( inf );

{\*用户键入编译源程序文件名\*}

assign( psin, inf );

reset( psin );

write( 'Source listing file ?');

readln( outf );

{\*用户键入编译输出程序文件名\*}

assign( psout, outf );

rewrite( psout );

assign ( prd, 'con' );

write( 'result file : ' );

readln( fprr );

assign( prr, fprr );

reset ( prd );

rewrite( prr );

{\*以下对声明的变量逐一进行初始化\*}

t := -1; {\*设置tab表的栈顶初值,以下类似, 设置各自的索引变量\*}

a := 0; {\*atab表\*}

b := 1; {\*btab表\*}

sx := 0; {\*stab表\*}

c2 := 0; {\*rconst表\*}

display[0] := 1; {\*设置display的初值,第一层\*}

iflag := false; {\*一系列flag的初始化\*}

oflag := false;

skipflag := false;

prtables := false;

stackdump := false;

insymbol; {\*预读入一个符号\*}

if sy <> programsy {\*不是以program开始程序错误\*}

then error(3)

else begin

insymbol;

if sy <> ident{\*不是标识符错误\*}

then error(2)

else begin

progname := id; {\*记录当前符号为程序名\*}

insymbol;

if sy <> lparent{\*下一符号不是左括号错误\*}

then error(9)

else repeat

insymbol; {\*循环读符号直到不是，\*}

if sy <> ident {\*不是标识符错误\*}

then error(2)

else begin

if id = 'input '

then iflag := true {\*读到INPUT\*}

else if id = 'output '

then oflag := true {\*读到OUTPUT\*}

else error(0); {\*标识符未定义错误\*}

insymbol

end

until sy <> comma; {\*预读符号后不是‘，’\*}

if sy = rparent {\*是右括号则继续读符号\*}

then insymbol

else error(4); {\*右括号不匹配错误\*}

if not oflag then error(20) {\*程序缺少参数OUTPUT\*}

end

end;

enterids; {\*在符号表中登记标准类型、函数和过程的名字以及它们的相应信息\*}

with btab[1] do {\*对分程序表中第一项的各个域进行初始化\*}

begin

last := t;

lastpar := 1;

psize := 0;

vsize := 0;

end;

block( blockbegsys + statbegsys, false, 1 );

{\*分析处理分程序,进入block分析(最外层)\*}

{\*main编译启动入口 从最外层的block开始分析\*}

if sy <> period {\*符号不是标识符错误\*}

then error(2);

emit(31); { halt } {\*生成fct = 31 的 P-code指令\*}

if prtables {\*如果打印表标志位为真则调用打印表子程序\*}

then printtables;

if errs = [] {\*若错误表为空则调用解释执行程序\*}

then interpret

else begin {\*否则打印错误信息\*}

writeln( psout );

writeln( psout, 'compiled with errors' );

writeln( psout );

errormsg;{\*打印错误信息 \*}

end;

writeln( psout );

close( psout ); {\*关闭文件\*}

close( prr )

end.

# 体会与感想

本次实验让我对未来我们即将实现的编译器有了进一步的了解，也加深了我对理论课所学习有关编译的技术的理解。同时通过为源代码编写注释的方式，让我对编译器设计的基本思想加深了印象。

本次实验共耗时在168小时以上，整体难度适中。课程提供了良好的指导，指导书相关要求与解释明确，起到了良好的引导作用。但本人仍经历了大量的摸索与尝试，经过数次尝试运行过程，最终才得以完成本次实验。

希望在后续的实验中能有更为清晰易懂的指导书和老师、助教的指导来指导我们更高效的完成实验，也希望理论课能增加相关与实验联系更为紧密的章节进行讲解说明。

另外，希望能够提供有关于实验题目和完成情况的反馈，以提高学习效果。

**陈麒先**

2018/11/7