附录A PL/0编译系统源代码

program pl0 ; { version 1.0 oct.1989 }

{ PL/0 compiler with code generation }

const norw = 13; { no. of reserved words }

txmax = 100; { length of identifier table }

nmax = 14; { max. no. of digits in numbers }

al = 10; { length of identifiers }

amax = 2047; { maximum address }

levmax = 3; { maximum depth of block nesting }

cxmax = 200; { size of code array }

type symbol =

( nul,ident,number,plus,minus,times,slash,oddsym,eql,neq,lss,

leq,gtr,geq,lparen,rparen,comma,semicolon,period,becomes,

beginsym,endsym,ifsym,thensym,whilesym,dosym,callsym,constsym,

varsym,procsym,readsym,writesym ); {\*所有的symbol以助记符方式记录为枚举\*}

alfa = packed array[1..al] of char; {\*标识符名字类型\*}

objecttyp = (constant,variable,prosedure); {\*对象类型枚举\*}

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

fct = ( lit,opr,lod,sto,cal,int,jmp,jpc,red,wrt ); { functions }{\*Pcode指令枚举\*}

instruction = packed record

f : fct; { function code }{\*操作码\*}

l : 0..levmax; { level }{\*被引用的分程序与说明之间的层次差\*}

a : 0..amax; { displacement address }

end; {\*Pcode指令在程序中的中间存储形式\*}

{ lit 0, a : load constant a

opr 0, a : execute operation a

lod l, a : load variable l,a

sto l, a : store variable l,a

cal l, a : call procedure a at level l

int 0, a : increment t-register by a

jmp 0, a : jump to a

jpc 0, a : jump conditional to a

red l, a : read variable l,a

wrt 0, 0 : write stack-top

}

var ch : char; { last character read}{\*当前读入字符\*}

sym: symbol; { last symbol read }{\*当前symbol\*}

id : alfa; { last identifier read }{\*当前标识符名\*}

num: integer; { last number read }{\*当前常量值\*}

cc : integer; { character count }{\*当前读取字符在当前行位置\*}

ll : integer; { line length }{\*当前行长度\*}

kk,err: integer; {\*kk为上一个标识符名的长度，err为错误数\*}

cx : integer; { code allocation index }{\*当前生成指令指针\*}

line: array[1..81] of char; {\*缓冲一行代码\*}

a : alfa; {\*当前读入的标识符名字缓存\*}

code : array[0..cxmax] of instruction; {\*Pcode指令储存数组\*}

word : array[1..norw] of alfa; {\*保留字表\*}

wsym : array[1..norw] of symbol; {\*保留字对应的symbol\*}

ssym : array[char] of symbol; {\*单个字符的symbol，一个字符一个symbol\*}

mnemonic : array[fct] of

packed array[1..5] of char; {\*输出Pcode指令时，每一种指令对应要输出的字符串\*}

declbegsys, statbegsys, facbegsys : symset; {\*三个头符号集\*}

table : array[0..txmax] of

record

name : alfa; {\*标识名字\*}

case kind: objecttyp of {\*标识类型\*}

constant : (val:integer );

variable,prosedure: (level,adr: integer )

end; {\*符号表\*}

fin : text; { source program file }

sfile: string; { source program file name }

procedure error ( n : integer ); {\*错误处理过程\*}

begin

writeln( '\*\*\*\*', ' ':cc-1, '^', n:2 ); {\*输出错误信息，'^'指向出错位置，n表示错误类型\*}

err := err+1 {\*错误数加1\*}

end; { error }

procedure getsym; {\*词法分析，读取一个symbol\*}

var i,j,k : integer; {\*定义临时变量\*}

procedure getch; {\*读下一个字符\*}

begin

if cc = ll { get character to end of line }{\*读到达当前行末\*}

then begin { read next line }{\*将下一行存入line\*}

if eof(fin) {\*若文件结束\*}

then begin

writeln('program incomplete'); {\*程序不完整\*}

close(fin); {\*关闭fin\*}

exit; {\*退出编译程序\*}

end;

ll := 0; {\*行长度置0\*}

cc := 0; {\*当前字符位置置0\*}

write(cx:4,' '); { print code address } {\*输出当前指令号\*}

while not eoln(fin) do {\*将源文件当前行存到行缓冲\*}

begin

ll := ll+1; {\*行长度加1\*}

read(fin,ch); {\*读一个字符\*}

write(ch); {\*输出这个字符\*}

line[ll] := ch {\*将这个字符放入行缓冲\*}

end;

writeln; {\*换行\*}

readln(fin); {\*读换行\*}

ll := ll+1; {\*行长度加1\*}

line[ll] := ' ' { process end-line }{\*行缓冲末置为空\*}

end;

cc := cc+1; {\*当前读取字符位置加1\*}

ch := line[cc] {\*读取当前字符\*}

end; { getch }

begin { procedure getsym; }

while ch = ' ' do {\*跳过空白符\*}

getch; {\*读下一个字符\*}

if ch in ['a'..'z'] {\*如果读入的字符为字母\*}

then begin { identifier of reserved word }

k := 0; {\*初始化读入标识符长度为0\*}

repeat {\*读取完整标识符名\*}

if k < al {\*标识符长度需小于最大标识符长度限制al，若大于al只取前al个字符\*}

then begin

k := k+1; {\*标识符长度加1\*}

a[k] := ch {\*读入的字符添加到标识符名\*}

end;

getch {\*读下一个字符\*}

until not( ch in ['a'..'z','0'..'9']); {\*直到下一个字符不是字母或数字\*}

if k >= kk { kk : last identifier length }

then kk := k

else repeat {\*清除掉上一个标识符名在a中的遗留\*}

a[kk] := ' ';

kk := kk-1

until kk = k;

id := a; {\*将读取的标识符名存入id\*}

i := 1; {\*i,j分别为二分查找顶、底下标\*}

j := norw; { binary search reserved word table }

repeat {\*查找id是否为保留字\*}

k := (i+j) div 2; {\*k取中间\*}

if id <= word[k] {\*id比第k个word小\*}

then j := k-1; {\*置顶下标为k-1\*}

if id >= word[k] {\*id比第k个word大\*}

then i := k+1 {\*置底下标为k+1\*}

until i > j; {\*底下标比顶下标大结束查找\*}

if i-1 > j

then sym := wsym[k] {\*是保留字，将sym置为此保留字对应symbol\*}

else sym := ident {\*非保留字，将sym置为标识符\*}

end

else if ch in ['0'..'9'] {\*如果读入的字符为数字\*}

then begin { number }

k := 0; {\*初始化读入数字长度为0\*}

num := 0; {\*初始化数字的数值为0\*}

sym := number; {\*置sym为数字\*}

repeat

num := 10\*num+(ord(ch)-ord('0')); {\*更新数值\*}

k := k+1; {\*读入数字长度加1\*}

getch {\*读下一个字符\*}

until not( ch in ['0'..'9']); {\*读到下一个非数字为止\*}

if k > nmax {\*读入数字长度超过最大限制\*}

then error(30) {\*报30号错误\*}

end

else if ch = ':' {\*如果读入的字符为':'\*}

then begin

getch; {\*读下一个字符\*}

if ch = '=' {\*如果读入的字符为'='\*}

then begin

sym := becomes; {\*置sym为赋值号\*}

getch {\*读下一个字符\*}

end

else sym := nul {\*置sym为空\*}

end

else if ch = '<' {\*如果读入的字符为'<'\*}

then begin

getch; {\*读下一个字符\*}

if ch = '=' {\*如果读入的字符为'='\*}

then begin

sym := leq; {\*置sym为小于等于号\*}

getch {\*读下一个字符\*}

end

else if ch = '>' {\*如果读入的字符为'>'\*}

then begin

sym := neq; {\*置sym为不等号\*}

getch {\*读下一个字符\*}

end

else sym := lss {\*置sym为小于号\*}

end

else if ch = '>' {\*如果读入的字符为'>'\*}

then begin

getch; {\*读下一个字符\*}

if ch = '=' {\*如果读入的字符为'='\*}

then begin

sym := geq; {\*置sym为大于等于号\*}

getch {\*读下一个字符\*}

end

else sym := gtr {\*置sym为大于号\*}

end

else begin {\*如果读入为其他字符\*}

sym := ssym[ch]; {\*将sym置为此字符对应的symbol\*}

getch {\*读下一个字符\*}

end

end; { getsym }

procedure gen( x: fct; y,z : integer );

{\*生成一个Pcode指令并存入code中\*}

begin

if cx > cxmax {\*若当前指令号大于最大指令数现在\*}

then begin {\*报错并退出\*}

writeln('program too long');

close(fin);

exit

end;

with code[cx] do

begin {\*对解析好的当前行代码生成Pcode指令存入code\*}

f := x;

l := y;

a := z

end;

cx := cx+1 {\*指令位置加1\*}

end; { gen }

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

{\*测试当前符号是否为当前语法成分合法后继，若不合法进行跳读

s1是合法后继符号集，s2是其他可以停止跳读的符号集合\*}

begin

if not ( sym in s1 ) {\*若不是合法的语法成分的后继符号，报错\*}

then begin

error(n); {\*报n号错误\*}

s1 := s1+s2; {\*将合法符号和停止符号合并\*}

while not( sym in s1) do {\*进行跳读，错误处理时，读取到下一个合法符号后继续编译\*}

getsym {\*读取下一个symbol\*}

end

end; { test }

procedure block( lev,tx : integer; fsys : symset );

{\*分程序分析处理，lev为分程序的嵌套层数，tx为符号表指针，fsys为后继符号集\*}

var dx : integer; { data allocation index }

tx0: integer; { initial table index }

cx0: integer; { initial code index }

procedure enter( k : objecttyp );

{\*将常量、变量或过程登记符入号表，k表示被登记的类型\*}

begin { enter object into table }

tx := tx+1; {\*符号表指针加1\*}

with table[tx] do {\*登记符号表\*}

begin

name := id; {\*登记标识符名\*}

kind := k; {\*登记类型\*}

case k of

constant : begin {\*如果是常量\*}

if num > amax {\*值超过最大限制\*}

then begin

error(30); {\*报30号错误\*}

num := 0 {\*值置为0\*}

end;

val := num {\*登记常量值\*}

end;

variable : begin {\*如果是变量\*}

level := lev; {\*登记嵌套层数\*}

adr := dx; {\*登记地址为当前层变量待登记地址\*}

dx := dx+1 {\*当前层变量待地址加1\*}

end;

prosedure: level := lev; {\*如果是过程，登记嵌套层数\*}

end

end

end; { enter }

function position ( id : alfa ): integer;

{\*查找标识符是否在符号表，id为标识符名\*}

var i : integer;

begin

table[0].name := id;

i := tx; {\*自顶向下查找符号表，若找不到返回0\*}

while table[i].name <> id do

i := i-1;

position := i {\*找到返回标识符位置\*}

end; { position }

procedure constdeclaration;

{\*常量声明处理\*}

begin

if sym = ident {\*若当前符号为标识符\*}

then begin

getsym; {\*读取下一个symbol\*}

if sym in [eql,becomes] {\*当前符号为等号或赋值号\*}

then begin

if sym = becomes {\*如果当前符号是等号\*}

then error(1); {\*报1号错误\*}

getsym; {\*读取下一个symbol\*}

if sym = number {\*如果当前符号是数字\*}

then begin

enter(constant); {\*登记这个常量\*}

getsym {\*读取下一个symbol\*}

end

else error(2) {\*报2号错误\*}

end

else error(3) {\*标识符后不是等号，报3号错误\*}

end

else error(4) {\*当前符号不为标识符，报4号错误\*}

end; { constdeclaration }

procedure vardeclaration;

{\*变量声明处理\*}

begin

if sym = ident {\*若当前符号为标识符\*}

then begin

enter(variable); {\*登记此变量\*}

getsym {\*读取下一个symbol\*}

end

else error(4) {\*报4号错误\*}

end; { vardeclaration }

procedure listcode;

{\*打印输出当前分程序生成的Pcode代码\*}

var i : integer;

begin

for i := cx0 to cx-1 do {\*cx0为当前分程序的起始行，cx为当前行（终止行）\*}

with code[i] do

writeln( i:4, mnemonic[f]:7,l:3, a:5) {\*将储存在code中的Pcode指令打印输出\*}

end; { listcode }

procedure statement( fsys : symset );

{\*语句分析处理，这里及下面的fsys都是相应块的后继符号集\*}

var i,cx1,cx2: integer;

procedure expression( fsys: symset);

{\*表达式分析处理\*}

var addop : symbol;

procedure term( fsys : symset);

{\*项分析处理\*}

var mulop: symbol ;

procedure factor( fsys : symset );

{\*因子分析处理\*}

var i : integer;

begin

test( facbegsys, fsys, 24 ); {\*测试头符号是否为因子头符号集\*}

while sym in facbegsys do {\*使得因子后紧接着一个因子的错误（漏掉乘除号）能继续编译\*}

begin

if sym = ident {\*若当前符号为标识符\*}

then begin

i := position(id); {\*查找符号表\*}

if i= 0 {\*若不在符号表中\*}

then error(11) {\*报11号错误\*}

else {\*在符号表中\*}

with table[i] do

case kind of

constant : gen(lit,0,val); {\*生成Pcode指令，载入常量的常数值\*}

variable : gen(lod,lev-level,adr); {\*生成Pcode指令，载入变量\*}

prosedure: error(21) {\*报21号错误\*}

end;

getsym {\*读取下一个symbol\*}

end

else if sym = number {\*若当前符号为数字\*}

then begin

if num > amax {\*数值大于最大限制\*}

then begin

error(30); {\*报30号错误\*}

num := 0 {\*置数值为0\*}

end;

gen(lit,0,num); {\*生成Pcode指令，载入此数值代表的常数\*}

getsym {\*读取下一个symbol\*}

end

else if sym = lparen {\*若当前符号为左括号\*}

then begin

getsym; {\*读取下一个symbol\*}

expression([rparen]+fsys); {\*解析下一个语法项，表达式\*}

if sym = rparen {\*表达式解析完后，下一个符号是否为右括号\*}

then getsym {\*读取下一个symbol\*}

else error(22) {\*报22号错误\*}

end;

test(fsys,[lparen],23) {\*测试因子的后继符号集\*}

end

end; { factor }

begin { procedure term( fsys : symset);

var mulop: symbol ; }

factor( fsys+[times,slash]); {\*解析因子\*}

while sym in [times,slash] do {\*右递归，检查下一个是否为乘除号\*}

begin

mulop := sym;

getsym; {\*读取下一个symbol\*}

factor( fsys+[times,slash] );

if mulop = times {\*若为乘号\*}

then gen( opr,0,4 ) {\*生成Pcode指令，乘法指令\*}

else gen( opr,0,5) {\*生成Pcode指令，除法指令\*}

end

end; { term }

begin { procedure expression( fsys: symset);

var addop : symbol; }

if sym in [plus, minus] {\*第一个符号为正负号\*}

then begin

addop := sym; {\*记录下这个符号\*}

getsym; {\*读取下一个symbol\*}

term( fsys+[plus,minus]); {\*解析项\*}

if addop = minus {\*若前面符号为负号\*}

then gen(opr,0,1) {\*生成Pcode指令，取负\*}

end

else term( fsys+[plus,minus]); {\*解析项\*}

while sym in [plus,minus] do {\*右递归，检查下一个是否为加减号\*}

begin

addop := sym;

getsym; {\*读取下一个symbol\*}

term( fsys+[plus,minus] ); {\*解析项\*}

if addop = plus {\*若为加号\*}

then gen( opr,0,2) {\*生成Pcode指令，加法指令\*}

else gen( opr,0,3) {\*生成Pcode指令，减法指令\*}

end

end; { expression }

procedure condition( fsys : symset );

{\*条件语句分析处理\*}

var relop : symbol;

begin

if sym = oddsym {\*若当前符号为odd\*}

then begin

getsym; {\*读取下一个symbol\*}

expression(fsys); {\*解析后继的表达式\*}

gen(opr,0,6) {\*生成Pcode指令，判断是否为奇数\*}

end

else begin

expression( [eql,neq,lss,gtr,leq,geq]+fsys); {\*解析比较的第一个表达式\*}

if not( sym in [eql,neq,lss,leq,gtr,geq]) {\*中间若不是比较运算符\*}

then error(20) {\*报20号错误\*}

else begin

relop := sym; {\*记录比较运算符\*}

getsym; {\*读取下一个symbol\*}

expression(fsys); {\*解析比较的第二个表达式\*}

case relop of {\*根据比较运算符生成相应指令\*}

eql : gen(opr,0,8); {\*生成Pcode指令\*}

neq : gen(opr,0,9); {\*生成Pcode指令\*}

lss : gen(opr,0,10); {\*生成Pcode指令\*}

geq : gen(opr,0,11); {\*生成Pcode指令\*}

gtr : gen(opr,0,12); {\*生成Pcode指令\*}

leq : gen(opr,0,13); {\*生成Pcode指令\*}

end

end

end

end; { condition }

begin { procedure statement( fsys : symset );

var i,cx1,cx2: integer; }

if sym = ident {\*若当前symbol为标识符，表示是赋值语句\*}

then begin

i := position(id); {\*查找符号表\*}

if i= 0 {\*若不在符号表\*}

then error(11) {\*报11号错误\*}

else if table[i].kind <> variable {\*若不是变量\*}

then begin { giving value to non-variation }

error(12); {\*报12号错误\*}

i := 0

end;

getsym; {\*读取下一个symbol\*}

if sym = becomes {\*符号是否为赋值号\*}

then getsym {\*读取下一个symbol\*}

else error(13); {\*报13号错误\*}

expression(fsys); {\*分析赋值号后面的表达式\*}

if i <> 0 {\*赋值过程没有出错\*}

then

with table[i] do

gen(sto,lev-level,adr) {\*生成Pcode指令，将栈顶值存入变量的地址adr\*}

end

else if sym = callsym {\*若当前symbol为call，表示过程调用语句\*}

then begin

getsym; {\*读取下一个symbol\*}

if sym <> ident {\*若symbol不是标识符\*}

then error(14) {\*报14号错误\*}

else begin

i := position(id); {\*查找符号表\*}

if i = 0 {\*若不在符号表\*}

then error(11) {\*报11号错误\*}

else

with table[i] do

if kind = prosedure {\*判断call后面标识符是否为过程\*}

then gen(cal,lev-level,adr) {\*生成Pcode指令，调用过程\*}

else error(15); {\*报15号错误\*}

getsym {\*读取下一个symbol\*}

end

end

else if sym = ifsym {\*若当前symbol为if，表示条件语句\*}

then begin

getsym; {\*读取下一个symbol\*}

condition([thensym,dosym]+fsys); {\*解析关系运算\*}

if sym = thensym {\*下一个symbol需要为then\*}

then getsym {\*读取下一个symbol\*}

else error(16); {\*报16号错误\*}

cx1 := cx; {\*记录跳转指令的位置\*}

gen(jpc,0,0); {\*生成Pcode指令，条件跳转\*}

statement(fsys); {\*解析then后面的语句\*}

code[cx1].a := cx {\*将if的跳转指令的跳转位置设置为设置为整条if语句之后\*}

end

else if sym = beginsym {\*若当前symbol为begin，表示复合语句\*}

then begin

getsym; {\*读取下一个symbol\*}

statement([semicolon,endsym]+fsys); {\*解析语句\*}

while sym in ([semicolon]+statbegsys) do {\*右递归\*}

begin

if sym = semicolon {\*语句结尾是否为分号\*}

then getsym {\*读取下一个symbol\*}

else error(10); {\*报10号错误\*}

statement([semicolon,endsym]+fsys) {\*解析下一条语句\*}

end;

if sym = endsym {\*最后一个语法元素是否为end\*}

then getsym {\*读取下一个symbol\*}

else error(17) {\*报17号错误\*}

end

else if sym = whilesym {\*若当前symbol为while，表示当循环语句\*}

then begin

cx1 := cx; {\*记录循环开始的指令位置\*}

getsym; {\*读取下一个symbol\*}

condition([dosym]+fsys); {\*解析循环条件\*}

cx2 := cx; {\*记录条件跳转指令的位置\*}

gen(jpc,0,0); {\*生成Pcode指令，循环条件为假时跳出循环\*}

if sym = dosym {\*symbol需要为do\*}

then getsym {\*读取下一个symbol\*}

else error(18); {\*报18号错误\*}

statement(fsys); {\*解析循环的语句\*}

gen(jmp,0,cx1); {\*生成Pcode指令，一次循环结束后跳回循环头\*}

code[cx2].a := cx {\*将条件跳转指令的跳转位置设置为循环结束位置\*}

end

else if sym = readsym {\*若当前symbol为read，表示读语句\*}

then begin

getsym; {\*读取下一个symbol\*}

if sym = lparen {\*read下一个symbol需为左括号\*}

then

repeat

getsym; {\*读取下一个symbol\*}

if sym = ident {\*read内第一个symbol需要为标识符\*}

then begin

i := position(id); {\*差符号表\*}

if i = 0 {\*不在符号表\*}

then error(11) {\*报11号错误\*}

else if table[i].kind <> variable {\*标识符不是变量\*}

then begin

error(12); {\*报12号错误\*}

i := 0

end

else with table[i] do

gen(red,lev-level,adr) {\*生成Pcode指令，读指令\*}

end

else error(4); {\*报4号错误\*}

getsym; {\*读取下一个symbol\*}

until sym <> comma {\*右递归\*}

else error(40); {\*报40号错误\*}

if sym <> rparen {\*最后一个符号需为右括号\*}

then error(22); {\*报22号错误\*}

getsym {\*读取下一个symbol\*}

end

else if sym = writesym {\*若当前symbol为write，表示写语句\*}

then begin

getsym; {\*读取下一个symbol\*}

if sym = lparen {\*write下一个symbol需为左括号\*}

then begin

repeat

getsym; {\*读取下一个symbol\*}

expression([rparen,comma]+fsys); {\*解析读语句里面的表达式\*}

gen(wrt,0,0); {\*生成Pcode指令\*}

until sym <> comma; {\*右递归\*}

if sym <> rparen {\*最后一个符号需为右括号\*}

then error(22); {\*报22号错误\*}

getsym {\*读取下一个symbol\*}

end

else error(40) {\*报40号错误\*}

end;

test(fsys,[],19) {\*测试语句的后继符号集\*}

end; { statement }

begin { procedure block( lev,tx : integer; fsys : symset );

var dx : integer; /\* data allocation index \*/

tx0: integer; /\*initial table index \*/

cx0: integer; /\* initial code index \*/ }

dx := 3; {\*初始数据记录地址（前三个分别留给SL、DL、RA）\*}

tx0 := tx; {\*记录头符号表指针，其实对应的是这个分程序所在符号表中的记录位置\*}

table[tx].adr := cx; {\*将分程序开始地址记入符号表，开始地址的指令是下面生成的跳转指令\*}

gen(jmp,0,0); {\*生成Pcode指令，跳过声明部分，跳转到本分程序语句块部分

（中间会有过程声明，会生成一些Pcode指令，需要跳过）\*}

{ jump from declaration part to statement part }

if lev > levmax {\*若嵌套层数大于最大限制\*}

then error(32); {\*报32号错误\*}

repeat

if sym = constsym {\*若symbol为const，是常量声明部分\*}

then begin

getsym; {\*读取下一个symbol\*}

repeat

constdeclaration; {\*解析常量声明\*}

while sym = comma do {\*一个常量声明右递归（逗号隔开）\*}

begin

getsym; {\*读取下一个symbol\*}

constdeclaration {\*解析常量声明\*}

end;

if sym = semicolon {\*常量定义语句完后需要是分号\*}

then getsym {\*读取下一个symbol\*}

else error(5) {\*报5号错误\*}

until sym <> ident {\*常量定义语句右递归（分号隔开）\*}

end;

if sym = varsym {\*若symbol为var，是变量声明部分\*}

then begin

getsym; {\*读取下一个symbol\*}

repeat

vardeclaration; {\*解析变量声明\*}

while sym = comma do {\*一个变量声明右递归（逗号隔开）\*}

begin

getsym; {\*读取下一个symbol\*}

vardeclaration {\*解析变量声明\*}

end;

if sym = semicolon {\*变量声明语句完后需要是分号\*}

then getsym {\*读取下一个symbol\*}

else error(5) {\*报5号错误\*}

until sym <> ident; {\*变量声明语句右递归（分号隔开）\*}

end;

while sym = procsym do {\*若symbol为procedure，是过程声明部分\*}

begin

getsym; {\*读取下一个symbol\*}

if sym = ident {\*procedure保留字后需要是一个标识符\*}

then begin

enter(prosedure); {\*登记符号表\*}

getsym {\*读取下一个symbol\*}

end

else error(4); {\*报4号错误\*}

if sym = semicolon {\*过程声明语句完后需要是分号\*}

then getsym {\*读取下一个symbol\*}

else error(5); {\*报5号错误\*}

block(lev+1,tx,[semicolon]+fsys); {\*解析过程声明的分程序部分（过程块）\*}

{\*每次处理下一个子块时，传入的符号表指针都重新从tx开始

tx是局部的，表示当父块符号表指针，字块tx变化不影响父块

表明上一个子块处理完后，上一个子块的符号表都被清除了\*}

if sym = semicolon {\*语句结束后需是分号\*}

then begin

getsym; {\*读取下一个symbol\*}

test( statbegsys+[ident,procsym],fsys,6) {\*测试过程声明部分后继符号集是否满足\*}

end

else error(5) {\*报5号错误\*}

end;

test( statbegsys+[ident],declbegsys,7) {\*测试整个声明部分后继符号集是否满足\*}

until not ( sym in declbegsys ); {\*为了使声明顺序颠倒能继续编译\*}

code[table[tx0].adr].a := cx; {\*table[tx0].adr是分程序起始地址（一条跳转指令）

将此跳转指令的跳转地址设置为cx，即下面将要解析的语句部分的开始\*}

{ back enter statement code's start adr. }

with table[tx0] do

begin

adr := cx; { code's start address }{\*分程序实际起始地址是下面的语句部分的开始\*}

end;

cx0 := cx; {\*设置语句部分的起始地址（为了下面listcode时能跳过上面过程声明中产生的Pcode指令）\*}

gen(int,0,dx); {\*生成Pcode指令，分配运行栈空间\*} { topstack point to operation area }

statement( [semicolon,endsym]+fsys); {\*解析分程序的语句部分\*}

gen(opr,0,0); {\*生成Pcode指令，返回\*} { return }

test( fsys, [],8 ); {\*测试后继符号集是否满足\*}

listcode; {\*打印出当前分程序块所生成的所以Pcode指令\*}

end { block };

procedure interpret;

const stacksize = 500;

var p,b,t: integer; { program-,base-,topstack-register }

i : instruction;{ instruction register }

s : array[1..stacksize] of integer; { data store }{\*运行栈\*}

function base( l : integer ): integer;

{\*通过静态链SL求出数据区基地址，l为层次差\*}

var b1 : integer;

begin { find base l levels down }

b1 := b; {\*b为当前块基地址\*}

while l > 0 do

begin

b1 := s[b1]; {\*上一个块的基地址\*}

l := l-1 {\*迭代l次\*}

end;

base := b1 {\*返回要找的基地址\*}

end; { base }

begin

writeln( 'START PL/0' );

t := 0; {\*栈指针\*}

b := 1; {\*当前块基地址，只想当前块的直接外层基地址\*}

p := 0; {\*指令计数器\*}

s[1] := 0; {\*直接外层基地址\*}

s[2] := 0; {\*前驱块基地址\*}

s[3] := 0; {\*返回地址\*}

repeat

i := code[p]; {\*取指令\*}

p := p+1; {\*指令计数器加1\*}

with i do

case f of

lit : begin {\*取常数，将常数a放到栈顶\*}

t := t+1;

s[t]:= a;

end;

opr : case a of { operator }{\*数据操作运算\*}

0 : begin { return }{\*返回\*}

t := b-1; {\*将栈顶退到当前块基地址的上一个（即将当前块退栈）\*}

p := s[t+3]; {\*指令计数器置为返回地址\*}

b := s[t+2]; {\*当前块基地址置为前驱块基地址\*}

end;

1 : s[t] := -s[t]; {\*取负\*}

2 : begin {\*加法\*}

t := t-1;

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

end;

3 : begin {\*减法\*}

t := t-1;

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

end;

4 : begin {\*乘法\*}

t := t-1;

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

end;

5 : begin {\*除法\*}

t := t-1;

s[t] := s[t]div s[t+1]

end;

6 : s[t] := ord(odd(s[t])); {\*odd运算\*}

8 : begin {\*判断相等\*}

t := t-1;

s[t] := ord(s[t]=s[t+1])

end;

9 : begin {\*判断不等\*}

t := t-1;

s[t] := ord(s[t]<>s[t+1])

end;

10: begin {\*判断小于\*}

t := t-1;

s[t] := ord(s[t]< s[t+1])

end;

11: begin {\*判断大于等于\*}

t := t-1;

s[t] := ord(s[t] >= s[t+1])

end;

12: begin {\*判断大于\*}

t := t-1;

s[t] := ord(s[t] > s[t+1])

end;

13: begin {\*判断小于等于\*}

t := t-1;

s[t] := ord(s[t] <= s[t+1])

end;

end;

lod : begin {\*载入指令\*}

t := t+1;

s[t] := s[base(l)+a] {\*将前l层的第a个数载入栈顶\*}

end;

sto : begin {\*存储指令\*}

s[base(l)+a] := s[t]; { writeln(s[t]); }{\*将栈顶存入前l层的第a个数\*}

t := t-1

end;

cal : begin { generate new block mark }{\*调用指令，调用前l层的过程块\*}

s[t+1] := base(l);{\*设置直接外层地址\*}

s[t+2] := b; {\*前驱块基地址为当前基地址\*}

s[t+3] := p; {\*返回地址为当前程序计数器值\*}

b := t+1; {\*设置当前基地址\*}

p := a; {\*设置当前程序计数器\*}

end;

int : t := t+a; {\*栈顶加a\*}

jmp : p := a; {\*跳转到第a条指令\*}

jpc : begin {\*如果栈顶为0（false），跳转到第a条指令\*}

if s[t] = 0

then p := a;

t := t-1;

end;

red : begin {\*读操作\*}

writeln('??:');

readln(s[base(l)+a]);

end;

wrt : begin {\*写操作\*}

writeln(s[t]);

t := t+1

end

end { with,case }

until p = 0; {主程序返回地址为0，p=0说明主程序返回，整个程序执行结束}

writeln('END PL/0');

end; { interpret }

begin { main }

writeln('please input source program file name : '); {\*输出操作提示\*}

readln(sfile); {\*读取源文件文件名\*}

assign(fin,sfile); {\*加源文件名付给fin\*}

reset(fin); {\*打开源文件fin\*}

for ch := 'A' to ';' do

ssym[ch] := nul; {\*先将所有字符对应的symbol置为nul\*}

word[1] := 'begin '; word[2] := 'call ';

word[3] := 'const '; word[4] := 'do ';

word[5] := 'end '; word[6] := 'if ';

word[7] := 'odd '; word[8] := 'procedure ';

word[9] := 'read '; word[10]:= 'then ';

word[11]:= 'var '; word[12]:= 'while ';

word[13]:= 'write '; {\*初始化保留字表\*}

wsym[1] := beginsym; wsym[2] := callsym;

wsym[3] := constsym; wsym[4] := dosym;

wsym[5] := endsym; wsym[6] := ifsym;

wsym[7] := oddsym; wsym[8] := procsym;

wsym[9] := readsym; wsym[10]:= thensym;

wsym[11]:= varsym; wsym[12]:= whilesym;

wsym[13]:= writesym; {\*初始化保留字对应的symbol表\*}

ssym['+'] := plus; ssym['-'] := minus;

ssym['\*'] := times; ssym['/'] := slash;

ssym['('] := lparen; ssym[')'] := rparen;

ssym['='] := eql; ssym[','] := comma;

ssym['.'] := period;

ssym['<'] := lss; ssym['>'] := gtr;

ssym[';'] := semicolon; {\*赋给特殊字符其对应的symbol\*}

mnemonic[lit] := 'LIT '; mnemonic[opr] := 'OPR ';

mnemonic[lod] := 'LOD '; mnemonic[sto] := 'STO ';

mnemonic[cal] := 'CAL '; mnemonic[int] := 'INT ';

mnemonic[jmp] := 'JMP '; mnemonic[jpc] := 'JPC ';

mnemonic[red] := 'RED '; mnemonic[wrt] := 'WRT '; {\*初始化Pcode指令输出字符串表\*}

declbegsys := [ constsym, varsym, procsym ];

statbegsys := [ beginsym, callsym, ifsym, whilesym];

facbegsys := [ ident, number, lparen ]; {\*初始化因子头符号集\*}

err := 0; {\*错误数初始为0\*}

cc := 0; {\*当前读取字符位置初始为0\*}

cx := 0; {\*行号初始为0\*}

ll := 0; {\*行长度初始为0\*}

ch := ' '; {\*当前读入字符初始为空\*}

kk := al; {\*设置当前标识符名长度为最长（表明当前标识符名缓冲区占用满）\*}

getsym; {\*读取下一个symbol\*}

block( 0,0,[period]+declbegsys+statbegsys ); {\*进入分程序分析处理过程\*}

if sym <> period {\*最后一个字符不是'.'\*}

then error(9); {\*报9号错误\*}

if err = 0 {\*判断编译过程是否出错\*}

then interpret {\*没有出错，解释执行\*}

else write('ERRORS IN PL/0 PROGRAM'); {\*有出错，报告程序错误\*}

writeln; {\*换行\*}

close(fin) {\*关闭fin\*}

end.

**附录B Pascal-s编译系统源代码**

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 }

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

llng = 121; { input line length }

emax = 322; { max exponent of real numbers }

emin = -292; { min exponent }

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

tmax = 100; { size of table }

bmax = 20; { size of block-talbe }

amax = 30; { size of array-table }

c2max = 20; { size of real constant table }

csmax = 30; { max no. of cases }

cmax = 800; { size of code }

lmax = 7; { maximum level }

smax = 600; { size of string-table }

ermax = 58; { max error no. }

omax = 63; { highest order code }

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

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

lineleng = 132; { output line length }

linelimit = 200;

stacksize = 1450;

type

symbol = ( intcon, realcon, charcon, stringcon,

notsy, plus, minus, times, idiv, rdiv, imod, andsy, orsy,

eql, neq, gtr, geq, lss, leq,

lparent, rparent, lbrack, rbrack, comma, semicolon, period,

colon, becomes, constsy, typesy, varsy, funcsy,

procsy, arraysy, recordsy, programsy, ident,

beginsy, ifsy, casesy, repeatsy, whilesy, forsy,

endsy, elsesy, untilsy, ofsy, dosy, tosy, downtosy, thensy);

index = -xmax..+xmax;

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

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 }{\*包括常整数和字符常量\*}

sleng: integer; { string length }

cc: integer; { character counter }{\*当前读取字符在当前行位置\*}

lc: integer; { program location counter }{\*当前语句位置\*}

ll: integer; { length of current line }

errpos: integer; {\*上一个错误在当前行的位置，为0时表示尚无错误\*}

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;

id: alfa; { identifier from insymbol }

progname: alfa;

stantyps: typset;

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

line: array[1..llng] of char; {\*行缓冲\*}

key: array[1..nkw] of alfa; {\*保留字符号串集\*}

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

sps: array[char]of symbol; { special symbols }{\*特殊字符symbol\*}

display: array[0..lmax] of integer; {\*display区，记录从第一层到当前层所有层在btab中的位置\*}

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

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 }

packed record

inxtyp,eltyp: types;

elref,low,high,elsize,size: index

end;

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

packed record

last, lastpar, psize, vsize: index

end;

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

rconst: array[1..c2max] of real;

code: array[0..cmax] of order;

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

inf, outf, fprr: string;

procedure errormsg;

{\*输出整个编译过程中的错误信息摘要\*}

var k : integer;

msg: array[0..ermax] of alfa;

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);

writeln(psout,'key words');

k := 0;

while errs <> [] do

begin

while not( k in errs )do k := k + 1;

writeln(psout, k, ' ', msg[k] );

errs := errs - [k]

end { while errs }

end { errormsg } ;

procedure endskip;

{\*输出跳读信息\*}

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 }

{\*读取下一个字符\*}

begin

if cc = ll {\*读到了行末\*}

then begin

if eof( psin ) {\*源程序文件已到末尾，程序不完整\*}

then begin

writeln( psout );

writeln( psout, 'program incomplete' );

errormsg;

exit;

end;

if errpos <> 0

then begin

if skipflag then endskip;

writeln( psout );

errpos := 0

end;

write( psout, lc: 5, ' ');

ll := 0;

cc := 0;

while not eoln( psin ) do {\*读取一行至行缓冲\*}

begin

ll := ll + 1;

read( psin, ch );

write( psout, ch );

line[ll] := ch

end;

ll := ll + 1;

readln( psin );

line[ll] := ' ';

writeln( psout );

end;

cc := cc + 1;

ch := line[cc];

end { nextch };

procedure error( n: integer );

{\*输出错误信息及位置并添加错误信息到errs\*}

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 );

{\*表溢出错误处理\*}

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}

{\*读取下一个symbol\*}

label 1,2,3;

var i,j,k,e: integer;

procedure readscale;

{\*读取实常数科学计数法的指数值\*}

var s,sign: integer;

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' ))

then error( 40 )

else repeat

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

nextch;

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

e := s\*sign + e {\*在下面解释实数的地方，小数点每多1位e减1，rnum记录的实数是去掉小数点后的值

最后rnum\*10^e得到真实的实数值\*}

end { readscale };

procedure adjustscale;

{\*将科学计数法表示的实数转换为实数值\*}

var s : integer;

d, t : real;

begin

if k + e > emax {\*k是当前rnum记录的数的长度\*}

then error(21)

else if k + e < emin

then rnum := 0

else begin

s := abs(e);

t := 1.0;

d := 10.0;

repeat

while not odd(s) do

begin

s := s div 2;

d := sqr(d)

end;

s := s - 1;

t := d \* t

until s = 0;

if e >= 0

then rnum := rnum \* t

else rnum := rnum / t

end

end { adjustscale };

procedure options;

{\*处理编译可选项，t+-表示控制编译完后是否打印各表，s+-表示是否打印运行栈\*}

{\*prtables变量控制是否打印表，stackdump控制是否打印运行栈\*}

procedure switch( var b: boolean );

begin

b := ch = '+';

if not b

then if not( ch = '-' )

then begin { print error message }

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

nextch;

end

else nextch

else nextch

end { switch };

begin { options }

repeat

nextch;

if ch <> '\*'

then begin

if ch = 't'

then begin

nextch;

switch( prtables )

end

else if ch = 's'

then begin

nextch;

switch( stackdump )

end;

end

until ch <> ','

end { options };

begin { insymbol }

1: while( ch = ' ' ) or ( ch = chr(9) ) do {\*跳过空白符\*}

nextch; { space & htab }

case ch of {\*对读到的字符分类\*}

'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': {\*第一个符号是字母，则下一个symbol可能是标识符或保留字\*}

begin { identifier of wordsymbol }

k := 0;

id := ' ';

repeat

if k < alng

then begin

k := k + 1;

id[k] := ch

end;

nextch

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

i := 1;

j := nkw; { binary search }

repeat

k := ( i + j ) div 2;

if id <= key[k]

then j := k - 1;

if id >= key[k]

then i := k + 1;

until i > j;

if i - 1 > j

then sy := ksy[k]

else sy := ident

end;

'0','1','2','3','4','5','6','7','8','9': {\*第一个symbol是数字，则下一个symbol可能是整数或实数\*}

begin { number }

k := 0;

inum := 0;

sy := intcon;

repeat

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

k := k + 1;

nextch

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

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

then begin

error(21);

inum := 0;

k := 0

end;

if ch = '.'

then begin

nextch;

if ch = '.'

then ch := ':' {\*两个点相当于冒号\*}

else begin

sy := realcon;

rnum := inum;

e := 0;

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

begin

e := e - 1;

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

nextch

end;

if e = 0

then error(40);

if ch = 'e'

then readscale;

if e <> 0 then adjustscale

end

end

else if ch = 'e'

then begin

sy := realcon;

rnum := inum;

e := 0;

readscale;

if e <> 0

then adjustscale

end;

end;

':': {\*冒号，则下一个symbol可能是赋值号或冒号\*}

begin

nextch;

if ch = '='

then begin

sy := becomes;

nextch

end

else sy := colon

end;

'<': {\*小于号，则下一个symbol可能是小于等于、不等于或小于\*}

begin

nextch;

if ch = '='

then begin

sy := leq;

nextch

end

else

if ch = '>'

then begin

sy := neq;

nextch

end

else sy := lss

end;

'>': {\*大于号，则下一个symbol可能是大于等于、大于\*}

begin

nextch;

if ch = '='

then begin

sy := geq;

nextch

end

else sy := gtr

end;

'.': {\*句号，则下一个symbol可能是句号或冒号（两个点相当于冒号）\*}

begin

nextch;

if ch = '.'

then begin

sy := colon;

nextch

end

else sy := period

end;

'''': {\*引号，则下一个symbol可能是字符或字符串\*}

begin

k := 0;

2:nextch; {\*读取引号里的一个字符\*}

if ch = ''''{\*读到一个引号\*}

then begin

nextch;

if ch <> ''''{\*下一个不是引号，说明字符串结束（否则代表一个引号）\*}

then goto 3

end;

if sx + k = smax

then fatal(7);

stab[sx+k] := ch;

k := k + 1;

if cc = 1 {\*中途换行\*}

then begin { end of line }

k := 0;

end

else goto 2;

3:if k = 1{\*k=0为空串或中途换行,k=1为字符,k>1为字符串\*}

then begin

sy := charcon;

inum := ord( stab[sx] )

end

else if k = 0

then begin

error(38);

sy := charcon;

inum := 0

end

else begin

sy := stringcon;

inum := sx;

sleng := k;

sx := sx + k

end

end;

'(': {\*左括号，则下一个symbol可能是左括号或者后面是编译可选项\*}

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;

'{': {\*左大括号，则后面是注释\*}

begin

nextch;

if ch = '$'

then options;

while ch <> '}' do

nextch;

nextch;

goto 1

end;

'+', '-', '\*', '/', ')', '=', ',', '[', ']', ';': {\*特殊符号，则下一个symbol为特殊符号对应symbol\*}

begin

sy := sps[ch];

nextch

end;

'$','"' ,'@', '?', '&', '^', '!': {\*非法符号\*}

begin

error(24);

nextch;

goto 1

end

end { case }

end { insymbol };

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

{\*登记符号表，x0为符号名，x1为种类（常量变量），x2为类型（int,char），x3是地址或值（登记入adr）\*}

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 );

{\*登记数组信息表，tp为下标类型，l、h分别为下、上界\*}

begin

{\*a为当前数组信息表顶部指针\*}

if l > h

then error(27);

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

then begin

error(27);

l := 0;

h := 0;

end;

if a = amax

then fatal(4)

else begin

a := a + 1;

with atab[a] do

begin

inxtyp := tp;

low := l;

high := h

end

end

end { enterarray };

procedure enterblock;

{\*登记分程序表（实际只是分配一个分程序表中的空间）\*}

begin

{\*b为当前分程序表顶部指针\*}

if b = bmax

then fatal(2)

else begin

b := b + 1;

btab[b].last := 0;

btab[b].lastpar := 0;

end

end { enterblock };

procedure enterreal( x: real );

{\*登记实常量表，x为待登记的实常数\*}

begin

{\*c1记录x在表中位置，c2为当前实常数表顶部指针\*}

if c2 = c2max - 1

then fatal(3)

else begin

rconst[c2+1] := x;

c1 := 1;

while rconst[c1] <> x do {\*查找是否已登记，已登记则更新表\*}

c1 := c1 + 1;

if c1 > c2

then c2 := c1

end

end { enterreal };

procedure emit( fct: integer );

{\*生成Pcode指令（无操作数）\*}

begin

if lc = cmax

then fatal(6);

code[lc].f := fct;

lc := lc + 1

end { emit };

procedure emit1( fct, b: integer );

{\*生成Pcode指令（1个操作数）\*}

begin

if lc = cmax

then fatal(6);

with code[lc] do

begin

f := fct;

y := b;

end;

lc := lc + 1

end { emit1 };

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

{\*生成Pcode指令（2个操作数）\*}

begin

if lc = cmax then fatal(6);

with code[lc] do

begin

f := fct;

x := a;

y := b

end;

lc := lc + 1;

end { emit2 };

procedure printtables;

{\*打印编译完后的符号表、分程序表、数组信息表和Pcode指令\*}

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 ';

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 {\*打印符号表（除去前28位保留的）\*}

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 {\*打印分程序表\*}

with btab[i] do

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

writeln( psout );

writeln( psout );

writeln( psout );

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

writeln( psout );

for i := 1 to a do {\*打印数组信息表\*}

with atab[i] do

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

writeln( psout );

writeln( psout );

writeln( psout );

writeln( psout, 'code:');

writeln( psout );

for i := 0 to lc-1 do {\*打印Pcode指令\*}

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 );

{\*解析分程序，fsys为后继符号集，isfun表示分程序是否为函数，level表示嵌套层数\*}

type conrec = record

case tp: types of

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

reals :( r:real )

end; {\*常量类，保存常量的类型和值\*}

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);

{\*跳读\*}

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 );

{\*测试sy是否在符号集s1\*}

begin

if not( sy in s1 )

then skip( s1 + s2, n )

end { test };

procedure testsemicolon;

{\*测试是否为分号\*}

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 );

{\*登记id表示的标识符于符号表\*}

var j,l : integer;

begin

if t = tmax

then fatal(1)

else begin

tab[0].name := id;

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

l := j; {\*l、j是当前块在符号表最后一个登记的位置，l用于下面记录的link用\*}

while tab[j].name <> id do {\*查找待登记标识符是否在符号表中\*}

j := tab[j].link;

if j <> 0 {\*在符号表中，报错重复定义\*}

then error(1)

else begin

t := t + 1;

with tab[t] do

begin

name := id;

link := l;

obj := k;

typ := notyp;

ref := 0;

lev := level;

adr := 0;

normal := false { initial value }

end;

btab[display[level]].last := t {\*更新前块在符号表最后一个记录的位置\*}

end

end

end { enter };

function loc( id: alfa ):integer;

{\*查找id表示的标识符是否在符号表合法位置（即在符号表中而且是其祖先块）\*}

{\*找到返回在符号表中的位置，找不到返回0\*}

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

begin

i := level;

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

repeat

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

while tab[j].name <> id do

j := tab[j].link;

i := i - 1;

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

if j = 0

then error(0);

loc := j

end { loc } ;

procedure entervariable;

{\*登记当前id表示的变量于符号表\*}

begin

if sy = ident

then begin

enter( id, vvariable );

insymbol

end

else error(2)

end { entervariable };

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

{\*常量解析，得到的常量保存在c\*}

var x, sign : integer;

begin

c.tp := notyp;

c.i := 0;

test( constbegsys, fsys, 50 );

if sy in constbegsys

then begin

if sy = charcon {\*字符常量\*}

then begin

c.tp := chars;

c.i := inum;

insymbol

end

else begin

sign := 1;

if sy in [plus, minus]

then begin

if sy = minus

then sign := -1;

insymbol

end;

if sy = ident {\*标识符，可能是常量标识符\*}

then begin

x := loc(id);

if x <> 0

then

if tab[x].obj <> konstant

then error(25)

else begin

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

if c.tp = reals

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

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

end;

insymbol

end

else if sy = intcon {\*整常数\*}

then begin

c.tp := ints;

c.i := sign\*inum;

insymbol

end

else if sy = realcon {\*实常数\*}

then begin

c.tp := reals;

c.r := sign\*rnum;

insymbol

end

else skip(fsys,50)

end;

test(fsys,[],6)

end

end { constant };

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

{\*解析一个类型（可能是自定义的、保留的、数组或record）\*}

{\*tp返回该类型最终的真实类型（保留的、数组或record），rf为引用信息，sz为此类型的占用大小\*}

var eltp : types;

elrf, x : integer;

elsz, offset, t0, t1 : integer;

procedure arraytyp( var aref, arsz: integer );

{\*解析数组类型，aref为解析完后该数组类型在数组信息表中的位置，arsz为数组大小\*}

var eltp : types;

low, high : conrec;

elrf, elsz: integer;

begin

constant( [colon, rbrack, rparent, ofsy] + fsys, low ); {\*解析数组下标\*}

if low.tp = reals {\*下标为实型，报错\*}

then begin

error(27);

low.tp := ints;

low.i := 0

end;

if sy = colon {\*‘..’符号\*}

then insymbol

else error(13);

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

if high.tp <> low.tp {\*上下标类型不同，报错\*}

then begin

error(27);

high.i := low.i

end;

enterarray( low.tp, low.i, high.i ); {\*将该数组登记入数组信息表\*}

aref := a; {\*记录在数组信息表中的位置\*}

if sy = comma {\*多维数组\*}

then begin

insymbol;

eltp := arrays; {\*数组元素为数组\*}

arraytyp( elrf, elsz ) {\*解析下一维\*}

end

else begin

if sy = rbrack {\*数组大小声明结束，symbol需要是右中括号，否则报错\*}

then insymbol

else begin

error(12);

if sy = rparent

then insymbol

end;

if sy = ofsy {\*中间有of关键字，否则报错\*}

then insymbol

else error(8);

typ( fsys, eltp, elrf, elsz ) {\*解析数组的单个元素的类型\*}

end;

with atab[aref] do

begin

arsz := (high-low+1) \* elsz; {\*需返回的数组大小\*}

size := arsz; {\*atab里面填入数组大小\*}

eltyp := eltp; {\*atab里面填入数组元素类型\*}

elref := elrf; {\*atab里面填入数组元素引用信息（在数组元素为数组或记录时）\*}

elsize := elsz {\*atab里面填入数组元素的大小\*}

end

end { arraytyp };

begin { typ }

tp := notyp;

rf := 0;

sz := 0;

test( typebegsys, fsys, 10 ); {\*测试头符号集\*}

if sy in typebegsys {\*判断当前symbol是否为标识符、数组array关键字或record关键字\*}

then begin

if sy = ident {\*当前symbol为标识符\*}

then begin

x := loc(id); {\*查找该标识符在符号表的位置\*}

if x <> 0 {\*在符号表中找到\*}

then with tab[x] do

if obj <> typel {\*该符号不是type类型，报错\*}

then error(29)

else begin {\*直接在符号表中取出要的信息\*}

tp := typ;

rf := ref;

sz := adr;

if tp = notyp

then error(30)

end;

insymbol

end

else if sy = arraysy {\*当前symbol为array关键字\*}

then begin

insymbol;

if sy = lbrack {\*左中括号\*}

then insymbol

else begin

error(11);

if sy = lparent

then insymbol

end;

tp := arrays; {\*tp设为为数组\*}

arraytyp(rf,sz) {\*rf和sz有arraytyp子过程给出\*}

end

else begin { records }{\*当前symbol为record关键字\*}

insymbol;

enterblock; {\*在btab中开一个空间给此record\*}

tp := records; {\*tp设为record类型\*}

rf := b; {\*引用地址rf设为在btab中的位置\*}

if level = lmax {\*若达到最大嵌套层数限制，报错\*}

then fatal(5);

level := level + 1; {\*嵌套层数加1\*}

display[level] := b; {\*在display区中记录下此层在btab中的位置\*}

offset := 0; {\*记录与record里面的变量在tab中的位置与此record在tab中位置的偏移（第一个变量偏移为0）\*}

while not ( sy in fsys - [semicolon,comma,ident]+ [endsy] ) do {\*读到record语句的末尾为止\*}

begin { field section }

if sy = ident {\*是一个标识符\*}

then begin

t0 := t; {\*t0记录一个类型下多个标识符的第一个\*}

entervariable; {\*记录这个record变量\*}

while sy = comma do {\*右递归，若为逗号继续读\*}

begin

insymbol;

entervariable

end;

if sy = colon {\*标识符名列举完毕，需要一个冒号，否则报错\*}

then insymbol

else error(5);

t1 := t; {\*t1记录一个类型下多个标识符的最后一个\*}

typ( fsys + [semicolon, endsy, comma,ident], eltp, elrf, elsz ); {\*解析这些变量的类型\*}

while t0 < t1 do {\*对这个类型下的所有标识符处理\*}

begin

t0 := t0 + 1;

with tab[t0] do {\*记录这个标识符的类型、引用、和偏移值\*}

begin

typ := eltp;

ref := elrf;

normal := true;

adr := offset;

offset := offset + elsz

end

end

end; { sy = ident }

if sy <> endsy {\*一个record变量后面不是end\*}

then begin

if sy = semicolon {\*也不是分号则报错，否则继续读\*}

then insymbol

else begin

error(14);

if sy = comma

then insymbol

end;

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

end

end; { field section }

btab[rf].vsize := offset; {\*设置vsize为总偏移\*}

sz := offset; {\*该记录类型的大小也是总偏移\*}

btab[rf].psize := 0;

insymbol;

level := level - 1 {\*退出这一层\*}

end; { record }

test( fsys, [],6 ) {\*测试后继符号集\*}

end;

end { typ };

procedure parameterlist; { formal parameter list }

{\*解析参数表并将形参登记入符号表\*}

var tp : types;

valpar : boolean;

rf, sz, x, t0 : integer;

begin

insymbol;

tp := notyp;

rf := 0;

sz := 0;

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

while sy in [ident, varsy] do {\*一直读到第一个符号不是标识符或者var关键词\*}

begin

if sy <> varsy {\*判断是否为可变形参\*}

then valpar := true {\*是传值形参\*}

else begin

insymbol;

valpar := false {\*可变形参\*}

end;

t0 := t; {\*t0记录同一类型形参的第一个\*}

entervariable;

while sy = comma do

begin

insymbol;

entervariable;

end;

if sy = colon {\*标识符名列举完毕之后，需要是一个冒号，否则报错\*}

then begin

insymbol;

if sy <> ident {\*需要一个代表类型名的标识符，否则报错\*}

then error(2)

else begin

x := loc(id); {\*查找这个类型在符号表的位置\*}

insymbol;

if x <> 0

then with tab[x] do

if obj <> typel {\*改标识符不是type类型，报错\*}

then error(29)

else begin

tp := typ;

rf := ref;

if valpar {\*如果是一个值引用形参，其大小为其类型大小，否则为1（记录地址即可）\*}

then sz := adr

else sz := 1

end;

end;

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

end

else error(5);

while t0 < t do {\*更改这个类型下所有形参的参数\*}

begin

t0 := t0 + 1;

with tab[t0] do

begin

typ := tp;

ref := rf;

adr := dx; {\*adr为在栈中的偏移量\*}

lev := level;

normal := valpar; {\*是否为传值形参\*}

dx := dx + sz {\*运行栈存入了此变量后栈顶指针变化\*}

end

end;

if sy <> rparent {\*声明完参数后需要是右括号\*}

then begin

if sy = semicolon {\*接着需要是分号\*}

then insymbol

else begin

error(14);

if sy = comma

then insymbol

end;

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

end

end { while };

if sy = rparent

then begin

insymbol;

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

end

else error(4)

end { parameterlist };

procedure constdec;

{\*常量声明解析\*}

var c : conrec;

begin

insymbol;

test([ident], blockbegsys, 2 );

while sy = ident do

begin

enter(id, konstant);

insymbol;

if sy = eql

then insymbol

else begin

error(16);

if sy = becomes

then insymbol

end;

constant([semicolon,comma,ident]+fsys,c); {\*取常量值\*}

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

tab[t].ref := 0;

if c.tp = reals {\*若为实数，登记实常量表，并登记此常量adr为在实常量表中的地址\*}

then begin

enterreal(c.r);

tab[t].adr := c1;

end

else tab[t].adr := c.i; {\*不是实数（整数、布尔或字符）直接登记adr为常量值\*}

testsemicolon

end

end { constdec };

procedure typedeclaration;

{\*类型声明解析\*}

var tp: types;

rf, sz, t1 : integer;

begin

insymbol;

test([ident], blockbegsys,2 );

while sy = ident do

begin

enter(id, typel); {\*先登记类型名\*}

t1 := t;

insymbol;

if sy = eql

then insymbol

else begin

error(16);

if sy = becomes

then insymbol

end;

typ( [semicolon,comma,ident]+fsys, tp,rf,sz ); {\*解析这个类型\*}

with tab[t1] do {\*登记具体的类型、在atab、btab或tab中的位置和此类型的大小\*}

begin

typ := tp;

ref := rf;

adr := sz

end;

testsemicolon

end

end { typedeclaration };

procedure variabledeclaration;

{\*变量声明解析\*}

var tp : types;

t0, t1, rf, sz : integer;

begin

insymbol;

while sy = ident do

begin

t0 := t; {\*t0为一个类型下第一个标识符在符号表中位置\*}

entervariable;

while sy = comma do

begin

insymbol;

entervariable;

end;

if sy = colon

then insymbol

else error(5);

t1 := t; {\*t1为一个类型下最后一个标识符在符号表中位置\*}

typ([semicolon,comma,ident]+fsys, tp,rf,sz ); {\*得到类型\*}

while t0 < t1 do

begin

t0 := t0 + 1;

with tab[t0] do

begin

typ := tp;

ref := rf;

lev := level;

adr := dx; {\*adr为将来改变量在运行栈中的地址\*}

normal := true;

dx := dx + sz {\*运行栈存入了此变量后栈顶指针变化\*}

end

end;

testsemicolon

end

end { variabledeclaration };

procedure procdeclaration;

{\*过程声明解析（只解析到过程/函数名），参数以后部分由block解析\*}

var isfun : boolean;

begin

isfun := sy = funcsy;

insymbol;

if sy <> ident

then begin

error(2);

id :=' '

end;

if isfun

then enter(id,funktion)

else enter(id,prozedure);

tab[t].normal := true;

insymbol;

block([semicolon]+fsys, isfun, level+1 ); {\*解析类型名后的块\*}

if sy = semicolon

then insymbol

else error(14);

emit(32+ord(isfun)) {exit}

end { proceduredeclaration };

procedure statement( fsys:symset );

{\*解析一个过程\*}

var i : integer;

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

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

{\*解析标识符选择数组下标或者选择record的域\*}

{\*v作为待解析的数组或record传入，并作为选中的数组元素或record域传出\*}

var x : item;

a,j : integer;

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

repeat

if sy = period {\*如果是句号，则是选择record的域\*}

then begin

insymbol; { field selector }

if sy <> ident {\*如果后面紧跟不是域的标识符名，报错\*}

then error(2)

else begin

if v.typ <> records {\*如果传入的v不是record类型，报错\*}

then error(31)

else begin { search field identifier }

j := btab[v.ref].last; {\*j先赋为该record在符号表中最后一个记录的位置\*}

tab[0].name := id;

while tab[j].name <> id do {\*查找该域名是否在record中\*}

j := tab[j].link;

if j = 0 {\*不在则报错\*}

then error(0);

v.typ := tab[j].typ; {\*将选中域的类型和引用赋值\*}

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

a := tab[j].adr; {\*该域相对于改record的相对偏移\*}

if a <> 0

then emit1(9,a) {\*现在栈顶指向record第一个域，加上a后使其指向选中域\*}

end;

insymbol

end

end

else begin { array selector }{\*是左括号，则是选择数组的元素\*}

if sy <> lbrack {\*不是左方括号，是左圆括号，报错\*}

then error(11);

repeat

insymbol;

expression( fsys+[comma,rbrack],x); {\*解析下标表达式\*}

if v.typ <> arrays {\*如果传入的v不是数组类型，报错\*}

then error(28)

else begin

a := v.ref; {\*将改数组引用在atab的地址赋给a\*}

if atab[a].inxtyp <> x.typ {\*如果在atab中存的下标类型与上面解析出来的下标类型不符，报错\*}

then error(26)

else if atab[a].elsize = 1

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

else emit1(21,a); {\*取下标变量地址（元素长度不为1）\*}

v.typ := atab[a].eltyp; {\*给选中元素赋值和引用\*}

v.ref := atab[a].elref

end

until sy <> comma; {\*多重数组后续下标\*}

if sy = rbrack {\*下标解析完后需要是右方括号，否则报错\*}

then insymbol

else begin

error(12);

if sy = rparent

then insymbol

end

end

until not( sy in[lbrack, lparent, period]); {解析多重数组或record嵌套}

test( fsys,[],6)

end { selector };

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

{\*处理自定义的函数或过程调用，i为函数或过程在符号表的位置\*}

var x : item;

lastp,cp,k : integer;

begin

emit1(18,i); { mark stack }

lastp := btab[tab[i].ref].lastpar; {\*最后一个参数在tab中的位置\*}

cp := i; {\*当前解析参数在tab的位置\*}

if sy = lparent

then begin { actual parameter list }{\*开始解析实参\*}

repeat

insymbol;

if cp >= lastp {\*解析过程中实参数超过了形参数，报错\*}

then error(39)

else begin

cp := cp + 1;

if tab[cp].normal

then begin { value parameter }{\*值传递的参数\*}

expression( fsys+[comma, colon,rparent],x); {\*解析实参表达式\*}

if x.typ = tab[cp].typ {\*形参和实参类型一样\*}

then begin

if x.ref <> tab[cp].ref {\*形参和实参引用地址不同\*}

then error(36) {\*报错（主要针对数组和record的情况，基本类型的引用都为0）\*}

else if x.typ = arrays {\*下面是数组和record的情况，需要装入整块数组或record\*}

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

else if x.typ = records

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

end

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

then emit1(26,0) {\*形参为实型，实参为整型，进行类型转换\*}

else if x.typ <> notyp {\*实参无类型，报错\*}

then error(36);

end

else begin { variable parameter }{\*传引用的参数\*}

if sy <> ident {\*传引用则实参必须是一个变量，否则报错\*}

then error(2) {\*实参不是一个标识符，报错\*}

else begin

k := loc(id); {\*在tab中找改标识符\*}

insymbol;

if k <> 0

then begin

if tab[k].obj <> vvariable {\*该标识符不是变量，报错\*}

then error(37);

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

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

if tab[k].normal {\*该变量本身是否为该层的传值形参\*}

then emit2(0,tab[k].lev,tab[k].adr) {\*是传值形参，则载入该变量地址\*}

else emit2(1,tab[k].lev,tab[k].adr); {\*否则载入该变量值\*}

if sy in [lbrack, lparent, period] {\*该变量有可能是数组或record\*}

then

selector(fsys+[comma,colon,rparent],x); {\*解析数组或record\*}

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

then error(36)

end

end

end {variable parameter }

end;

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

until sy <> comma;

if sy = rparent {\*参数解析完毕，需要是右括号，否则报错\*}

then insymbol

else error(4)

end;

if cp < lastp {\*实参数小于形参数，报错\*}

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

emit1(19,btab[tab[i].ref].psize-1 ); {\*过程调用指令，传入过程参数的大小\*}

if tab[i].lev < level {\*如果调用者与被调用者不同层，更新运行时的display区\*}

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

end { call };

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

{\*返回a和b运算得到的结果类型\*}

begin

{\*只有notyp和ints小于reals\*}

if ( a > reals ) or ( b > reals ) {\*a，b不是notyp，ints或reals，报错，返回notyp\*}

then begin

error(33);

resulttype := notyp

end

else if ( a = notyp ) or ( b = notyp ) {\*ab其中一个是notyp，返回notyp\*}

then resulttype := notyp

else if a = ints

then if b = ints {\*ab都是ints，返回ints\*}

then resulttype := ints

else begin {\*a是ints，b是reals，返回reals并转换a为reals\*}

resulttype := reals;

emit1(26,1)

end

else begin {\*a是reals则返回reals\*}

resulttype := reals;

if b = ints {\*a是reals，b是ints，将b转换为reals\*}

then emit1(26,0)

end

end { resulttype } ;

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

{\*解析一个表达式，参数x储存解析结果，储存了解析结果的类型和引用地址\*}

var y : item;

op : symbol;

{\*解析各个成分时，有很多地方允许了无类型不报错是为了防止重复报错\*}

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

{\*解析一个简单表达式，参数x储存解析结果\*}

var y : item;

op : symbol;

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

{\*解析一个项，参数x储存解析结果\*}

var y : item;

op : symbol;

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

{\*解析一个因子，参数x储存解析结果\*}

var i,f : integer;

procedure standfct( n: integer );

{\*解析标准函数，n为标准函数的号\*}

var ts : typset; {\*存放标准函数需要的参数类型\*}

begin { standard function no. n }

if sy = lparent {\*函数名后是一个左括号，否则报错\*}

then insymbol

else error(9);

if n < 17

then begin

expression( fsys+[rparent], x ); {\*解析标准函数的参数，以下标准函数都只有一个参数\*}

case n of {\*对标准函数的函数号进行分类讨论，并给出ts\*}

{\*下面有修改tab[i]的类型的，tab[i]是返回值类型。

是因为最后还要统一将x的类型置为函数的返回值类型。

而像abs，sqr这样的函数其返回值类型是由其参数决定的，因此要修改\*}

{ abs, sqr } 0,2: begin

ts := [ints, reals];

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

if x.typ = reals

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)

end;

end; { case }

if x.typ in ts {\*x的类型在对应函数的参数类型集合内\*}

then emit1(8,n) {\*生成调用标准函数的指令\*}

else if x.typ <> notyp {\*否则报错\*}

then error(48);

end

else begin { n in [17,18] }

{\*17，18分别是eof和eoln，检测是否是文件末尾或行末尾，参数为一个代表文件的标识符\*}

if sy <> ident

then error(2)

else if id <> 'input ' {\*如果参数名不是输入文件，报错\*}

then error(0)

else insymbol;

emit1(8,n); {\*生成调用标准函数的指令\*}

end;

x.typ := tab[i].typ; {\*将x类型设为函数返回值类型\*}

if sy = rparent {\*解析完参数后需要是右括号，否则报错\*}

then insymbol

else error(4)

end { standfct } ;

begin { factor }

x.typ := notyp;

x.ref := 0;

test( facbegsys, fsys,58 ); {\*测试因子头符号集\*}

while sy in facbegsys do

begin {\*对头符号分类\*}

if sy = ident {\*是一个标识符\*}

then begin

i := loc(id); {\*在符号表中找到该标识符\*}

insymbol;

with tab[i] do

case obj of {\*对该标识符类型进行分类\*}

konstant: begin {\*是常量\*}

x.typ := typ; {\*x类型置为这个常量的类型\*}

x.ref := 0;

if x.typ = reals {\*若该常量是实型\*}

then emit1(25,adr) {\*生成载入常量指令\*}

else emit1(24,adr) {\*否则生成载入字面常量指令\*}

end;

vvariable:begin {\*是变量\*}

x.typ := typ; {\*x类型为该变量类型\*}

x.ref := ref; {\*x引用为该变量引用\*}

if sy in [lbrack, lparent,period]

then begin {\*如果后面跟着左括号或点，则是数组或record\*}

if normal

then f := 0

else f := 1;

emit2(f,lev,adr); {\*载入这个变量\*}

selector(fsys,x); {\*选取数组或record的元素或域\*}

if x.typ in stantyps

then emit(34) {\*载入栈顶内容\*}

end

else begin

if x.typ in stantyps

then if normal

then f := 1

else f := 2

else if normal

then f := 0

else f := 1;

emit2(f,lev,adr) {\*载入这个变量\*}

end

end;

typel,prozedure: error(44); {\*是type类型或者过程，报错\*}

funktion: begin {\*是函数\*}

x.typ := typ; {\*因子类型置为函数返回值类型\*}

if lev <> 0 {\*若lev为0表示该函数是标准函数\*}

then call(fsys,i) {\*自定义的函数，调用call解析\*}

else standfct(adr) {\*标准函数，调用standfct解析\*}

end

end { case,with }

end

else if sy in [ charcon,intcon,realcon ] {\*是一个常数类型\*}

then begin

if sy = realcon {\*是一个实常数\*}

then begin

x.typ := reals; {\*x类型置为实型\*}

enterreal(rnum); {\*在实常量表中登记或者找到这个实常数\*}

emit1(25,c1) {\*生成指令装入c1位置的实数，c1为该实数在实常量表中的位置\*}

end

else begin {\*是整型或字符型\*}

if sy = charcon

then x.typ := chars {\*若是字符常量，x类型置为字符型\*}

else x.typ := ints; {\*否则是整数常量，x类型置为整型\*}

emit1(24,inum) {\*生成指令直接装入inum代表的数\*}

end;

x.ref := 0; {\*标准类型引用均为0\*}

insymbol

end

else if sy = lparent {\*是左括号\*}

then begin

insymbol;

expression(fsys + [rparent],x); {\*解析括号内的表达式\*}

if sy = rparent {\*表达式完后需要是右括号，否则报错\*}

then insymbol

else error(4)

end

else if sy = notsy {\*是not关键字\*}

then begin

insymbol;

factor(fsys,x); {\*解析not后面的因子\*}

if x.typ = bools {\*该因子是布尔型\*}

then emit(35) {\*生成逻辑非指令\*}

else if x.typ <> notyp {\*否则如果也不是无类型，报错\*}

then error(32)

end;

test(fsys,facbegsys,6) {\*测试因子后继符号集\*}

end { while }

end { factor };

begin { term }

{\*x保存解析到目前为止的解析结果\*}

factor( fsys + [times,rdiv,idiv,imod,andsy],x); {\*解析第一个因子\*}

while sy in [times,rdiv,idiv,imod,andsy] do {\*后面出现了操作符，则解析后续因子\*}

begin

op := sy; {\*op为操作符\*}

insymbol;

factor(fsys+[times,rdiv,idiv,imod,andsy],y ); {\*解析后续的因子，结果保存在y\*}

if op = times {\*乘法\*}

then begin

x.typ := resulttype(x.typ, y.typ); {\*获取xy运算后的类型\*}

case x.typ of

notyp: ;

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 {\*实数除法\*}

then begin

if x.typ = ints {\*x为整数，对x类型转换\*}

then begin

emit1(26,1);

x.typ := reals;

end;

if y.typ = ints {\*y为整数，对y类型转换\*}

then begin

emit1(26,0);

y.typ := reals;

end;

if (x.typ = reals) and (y.typ = reals) {\*xy均为实型\*}

then emit(61) {\*生成实数除指令\*}

else begin {\*否则报错\*}

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) {\*xy均为布尔型\*}

then emit(56) {\*生成与运算指令\*}

else begin {\*否则报错\*}

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

then error(32);

x.typ := notyp

end

end

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

if (x.typ = ints) and (y.typ = ints) {\*xy需要均为整型，否则报错\*}

then if op = idiv

then emit(58) {\*除法运算，生成整数除指令\*}

else emit(59) {\*模运算，生成取模指令\*}

else begin

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

then error(34);

x.typ := notyp

end

end

end { while }

end { term };

begin { simpleexpression }

{\*x保存解析到目前为止的解析结果\*}

if sy in [plus,minus] {\*第一个项前面有正负号\*}

then begin

op := sy;

insymbol;

term( fsys+[plus,minus],x); {\*解析第一个项\*}

if x.typ > reals {\*项不是数字类型，报错\*}

then error(33)

else if op = minus {\*前面有负号，生成取负指令\*}

then emit(36)

end

else term(fsys+[plus,minus,orsy],x); {\*第一个项前面没有正负号，直接解析\*}

while sy in [plus,minus,orsy] do {\*后面出现了操作符，继续解析后续项\*}

begin

op := sy; {\*op为操作符\*}

insymbol;

term(fsys+[plus,minus,orsy],y); {\*解析后续的一个项，结果存入y\*}

if op = orsy {\*如果是or\*}

then begin

if ( x.typ = bools )and(y.typ = bools) {\*or前后需要是布尔类型否则报错\*}

then emit(51) {\*生成逻辑或运算指令\*}

else begin

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

then error(32);

x.typ := notyp {\*or前后有不是布尔型的操作数，置x为无类型\*}

end

end

else begin

x.typ := resulttype(x.typ,y.typ); {\*解析结果类型为xy运算后的类型\*}

case x.typ of

notyp: ;

ints: if op = plus {\*结果为整数\*}

then emit(52) {\*整数加指令\*}

else emit(53); {\*整数减指令\*}

reals:if op = plus {\*结果为实数\*}

then emit(54) {\*实数加指令\*}

else emit(55) {\*实数减指令\*}

end { case }

end

end { while }

end { simpleexpression };

begin { expression }

simpleexpression(fsys+[eql,neq,lss,leq,gtr,geq],x); {\*解析一个简单表达式，结果存入x\*}

if sy in [ eql,neq,lss,leq,gtr,geq] {\*若后面跟着逻辑运算符，则是一个逻辑表达式\*}

then begin

op := sy; {\*op为操作符\*}

insymbol;

simpleexpression(fsys,y); {\*解析逻辑运算符后的简单表达式，结果存入y\*}

if(x.typ in [notyp,ints,bools,chars]) and (x.typ = y.typ) {\*xy类型要相同且为基本类型\*}

then case op of {\*生成逻辑运算的指令\*}

eql: emit(45);

neq: emit(46);

lss: emit(47);

leq: emit(48);

gtr: emit(49);

geq: emit(50);

end

else begin {\*下面检查xy是否一个为整型一个为实型，如果是进行强制类型转换再比较\*}

if x.typ = ints

then begin

x.typ := reals;

emit1(26,1)

end

else if y.typ = ints

then begin

y.typ := reals;

emit1(26,0)

end;

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

then case op of

eql: emit(39);

neq: emit(40);

lss: emit(41);

leq: emit(42);

gtr: emit(43);

geq: emit(44);

end

else error(35)

end;

x.typ := bools {\*结果类型为布尔型\*}

end

end { expression };

procedure assignment( lv, ad: integer );

{\*处理赋值语句，lv为待赋值变量所在层次，ad为待赋值变量在其层里的相对偏移\*}

var x,y: item;

f : integer;

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

{\*i为待赋值的标识符在tab中的指针\*}

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

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

if tab[i].normal

then f := 0

else f := 1;

emit2(f,lv,ad); {\*将变量地址装入栈顶\*}

if sy in [lbrack,lparent,period] {\*变量可能是数组或record\*}

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

if sy = becomes

then insymbol

else begin

error(51);

if sy = eql

then insymbol

end;

expression(fsys,y); {\*解析赋值的表达式\*}

if x.typ = y.typ

then if x.typ in stantyps {\*是标准类型，直接存入\*}

then emit(38)

else if x.ref <> y.ref

then error(46)

else if x.typ = arrays

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

else emit1(23,btab[x.ref].vsize) {\*record\*}

else if(x.typ = reals )and (y.typ = ints) {\*将整型转为实型\*}

then begin

emit1(26,0);

emit(38)

end

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

then error(46)

end { assignment };

procedure compoundstatement;

{\*处理复合语句\*}

begin

insymbol;

statement([semicolon,endsy]+fsys); {\*解析第一条语句\*}

while sy in [semicolon]+statbegsys do {\*解析后面的语句\*}

begin

if sy = semicolon

then insymbol

else error(14);

statement([semicolon,endsy]+fsys)

end;

if sy = endsy {\*最后一个symbol需要为end\*}

then insymbol

else error(57)

end { compoundstatement };

procedure ifstatement;

{\*处理if条件语句\*}

var x : item;

lc1,lc2: integer;

begin

insymbol;

expression( fsys+[thensy,dosy],x); {\*解析if条件\*}

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

then error(17);

lc1 := lc; {\*第一条跳转指令的地址\*}

emit(11); { jmpc }{\*if不成立时需要跳过then部分\*}

if sy = thensy

then insymbol

else begin

error(52);

if sy = dosy

then insymbol

end;

statement( fsys+[elsesy]);

if sy = elsesy

then begin

insymbol;

lc2 := lc; {\*第二条跳转指令地址\*}

emit(10); {\*if成立时需要跳过else部分\*}

code[lc1].y := lc; {\*将第一条跳转指令的跳转位置设置到else部分第一句\*}

statement(fsys);

code[lc2].y := lc {\*将第一条跳转指令的跳转位置设置到else最后\*}

end

else code[lc1].y := lc {\*没有else，将第一条跳转指令的跳转位置设置到if语句最后\*}

end { ifstatement };

procedure casestatement;

{\*处理case语句\*}

var x : item; {\*x为case选择表达式的解析结果\*}

i,j,k,lc1 : integer;

casetab : array[1..csmax]of

packed record

val,lc : index

end;{\*所以label的表，val为label值，lc存放这个label对应的指令位置\*}

exittab : array[1..csmax] of integer; {\*解析每个case分支的结束位置\*}

procedure caselabel;

{\*解析单个label\*}

var lab : conrec;

k : integer;

begin

constant( fsys+[comma,colon],lab ); {\*解析label常量，结果报存在lab\*}

if lab.tp <> x.typ {\*如果lab的类型和case选择表达式的类型不符，报错\*}

then error(47)

else if i = csmax {\*如果已达到最大的case数目，报错\*}

then fatal(6)

else begin

i := i+1;

k := 0;

casetab[i].val := lab.i; {\*将这个label填入case表\*}

casetab[i].lc := lc;

repeat {\*查找这个label在case表中是否出现\*}

k := k+1

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

if k < i {\*此label已经出现过\*}

then error(1); { multiple definition }

end

end { caselabel };

procedure onecase;

{\*解析一个case分支\*}

begin

if sy in constbegsys

then begin

caselabel; {\*解析第一个label\*}

while sy = comma do {\*解析后续的label\*}

begin

insymbol;

caselabel

end;

if sy = colon {\*label结束后需要是一个冒号，否则报错\*}

then insymbol

else error(5);

statement([semicolon,endsy]+fsys); {\*解析这个case分支的语句\*}

j := j+1;

exittab[j] := lc; {\*记录这个case的结束位置（指向后面生成的这个case的无条件跳转指令）\*}

emit(10)

end

end { onecase };

}

begin { casestatement }

insymbol;

i := 0; {\*i记录label数量\*}

j := 0; {\*j记录分支数量\*}

expression( fsys + [ofsy,comma,colon],x ); {\*解析选择表达式\*}

if not( x.typ in [ints,bools,chars,notyp ]) {\*如果选择表达式类型不在上述集合，报错\*}

then error(23);

lc1 := lc;

emit(12); {jmpx}{\*查找情况表进行跳转\*}

if sy = ofsy {\*需要是of关键字，否则报错\*}

then insymbol

else error(8);

onecase; {\*解析第一个分支\*}

while sy = semicolon do {\*解析后续分支\*}

begin

insymbol;

onecase

end;

code[lc1].y := lc;{\*设置情况表的位置\*}

for k := 1 to i do {\*将所有label登记入情况表\*}

begin

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

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

end;

emit1(10,0);{\*default？？？\*}

for k := 1 to j do

code[exittab[k]].y := lc; {\*将所有分支最后的跳转指令位置设置为整个case的结束位置\*}

if sy = endsy {\*最后需要是end关键字，否则报错\*}

then insymbol

else error(57)

end { casestatement };

procedure repeatstatement;

{\*解析repeat语句\*}

var x : item;

lc1: integer;

begin

lc1 := lc; {\*repeat开始时的指令地址\*}

insymbol;

statement( [semicolon,untilsy]+fsys); {\*解析repeat块内第一条语句\*}

while sy in [semicolon]+statbegsys do {\*解析repeat块内后续语句\*}

begin

if sy = semicolon

then insymbol

else error(14);

statement([semicolon,untilsy]+fsys)

end;

if sy = untilsy {\*语句结束后需要是until关键字，否则报错\*}

then begin

insymbol;

expression(fsys,x); {\*解析until条件\*}

if not(x.typ in [bools,notyp] ) {\*如果条件类型不是bools或无类型，报错\*}

then error(17);

emit1(11,lc1); {\*条件跳转到repeat开头\*}

end

else error(53)

end { repeatstatement };

procedure whilestatement;

{\*解析while语句\*}

var x : item;

lc1,lc2 : integer;

begin

insymbol;

lc1 := lc; {\*while开始时的指令地址\*}

expression( fsys+[dosy],x); {\*解析while条件\*}

if not( x.typ in [bools, notyp] ) {\*如果条件类型不是bools或无类型，报错\*}

then error(17);

lc2 := lc; {\*while跳转指令的地址\*}

emit(11); {\*while条件不成立时，跳到while结束部分\*}

if sy = dosy {\*后面需是do关键字，否则报错\*}

then insymbol

else error(54);

statement(fsys); {\*解析while的语句\*}

emit1(10,lc1); {\*while语句部分结束后，无条件跳到while开头判断循环条件\*}

code[lc2].y := lc {\*将循环条件不成立时，跳转的位置设置为while结束\*}

end { whilestatement };

procedure forstatement;

{\*解析for语句\*}

var cvt : types; {\*迭代变量的类型\*}

x : item;

i,f,lc1,lc2 : integer;

begin

insymbol;

if sy = ident {\*迭代变量的标识符\*}

then begin

i := loc(id); {\*查找迭代变量在tab的位置\*}

insymbol;

if i = 0 {\*如果迭代变量不在tab，将其设为整型\*}

then cvt := ints

else if tab[i].obj = vvariable {\*在符号表中，需要是一个变量类型，否则报错\*}

then begin

cvt := tab[i].typ;

if not tab[i].normal {\*如果迭代变量是个传值的形参，报错\*}

then error(37)

else emit2(0,tab[i].lev, tab[i].adr ); {\*载入这个变量作为迭代变量\*}

if not ( cvt in [notyp, ints, bools, chars]) {\*如果迭代变量的类型不是上述类型，报错\*}

then error(18)

end

else begin

error(37);

cvt := ints

end

end

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

if sy = becomes {\*后面是一个赋值号\*}

then begin

insymbol;

expression( [tosy, downtosy,dosy]+fsys,x); {\*解析表达式，是迭代变量的初始值\*}

if x.typ <> cvt

then error(19);

end

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

f := 14; {\*增量步长型的for语句\*}

if sy in [tosy,downtosy] {\*to或downto关键字\*}

then begin

if sy = downtosy

then f := 16; {\*如果关键字为downto，则是减量步长型\*}

insymbol;

expression([dosy]+fsys,x); {\*解析表达式，是迭代变量终止值\*}

if x.typ <> cvt

then error(19)

end

else skip([dosy]+fsys,55);

lc1 := lc; {\*for测试指令的地址\*}

emit(f); {\*for测试指令需要跳到for语句结尾\*}

if sy = dosy {\*do关键字\*}

then insymbol

else error(54);

lc2 := lc; {\*for中语句部分指令的开始地址\*}

statement(fsys); {\*解析语句部分\*}

emit1(f+1,lc2); {\*for语句结束后，生成再入指令\*}

code[lc1].y := lc {\*将for测试指令的跳转位置设为for语句结尾\*}

end { forstatement };

procedure standproc( n: integer );

{\*解析标准过程（read和write）\*}

var i,f : integer;

x,y : item;

begin

case n of

1,2 : begin { read }

if not iflag {\*若主程序参数中没有输入，报错\*}

then begin

error(20);

iflag := true

end;

if sy = lparent

then begin

repeat

insymbol;

if sy <> ident

then error(2)

else begin

i := loc(id); {\*查找读变量\*}

insymbol;

if i <> 0

then if tab[i].obj <> vvariable

then error(37)

else begin

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

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

if tab[i].normal

then f := 0

else f := 1;

emit2(f,tab[i].lev,tab[i].adr); {\*载入读变量\*}

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)) {\*读一个数\*}

else error(41)

end

end;

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

until sy <> comma;

if sy = rparent

then insymbol

else error(4)

end;

if n = 2 {\*如果是readln指令则读换行\*}

then emit(62)

end;

3,4 : begin { write }

if sy = lparent

then begin

repeat

insymbol;

if sy = stringcon {\*如果write括号内是字符串\*}

then begin

emit1(24,sleng); {\*装入sleng长度的字符常量\*}

emit1(28,inum); {\*写字符（inum此时指向被写字符串在stab中的起始位置）\*}

insymbol

end

else begin

expression(fsys+[comma,colon,rparent],x); {\*待写的表达式\*}

if not( x.typ in stantyps )

then error(41);

if sy = colon

then begin

insymbol;

expression( fsys+[comma,colon,rparent],y); {\*？？？\*}

if y.typ <> ints

then error(43);

if sy = colon

then begin

if x.typ <> reals

then error(42);

insymbol;

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

if y.typ <> ints

then error(43);

emit(37)

end

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

end

else emit1(29,ord(x.typ)) {\*写x的类型\*}

end

until sy <> comma;

if sy = rparent

then insymbol

else error(4)

end;

if n = 4 {\*如果是writeln，换行\*}

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) {\*非标准过程\*}

else standproc(tab[i].adr);{\*标准过程\*}

funktion: if tab[i].ref = display[level] {\*是函数的返回值\*}

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; {\*留出5个内务信息区\*}

prt := t; {\*此分程序在tab中的指针\*}

if level > lmax

then fatal(5);

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

enterblock; {\*申请btab空间\*}

prb := b; {\*此分程序在btab中的指针\*}

display[level] := b; {\*记录display区\*}

tab[prt].typ := notyp;

tab[prt].ref := prb; {\*记录过程在tab表中的引用\*}

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

then parameterlist;

btab[prb].lastpar := t; {\*记录最后一个参数在符号表的位置\*}

btab[prb].psize := dx; {\*记录最后一个参数在运行栈位置\*}

if isfun {\*若是函数，记录函数返回值类型\*}

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);

if sy = semicolon

then insymbol

else error(14);

repeat {\*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

procdeclaration;

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

until sy in statbegsys;

tab[prt].adr := lc; {\*分程序在tab的地址设为分程序中语句部分第一个指令的地址\*}

insymbol;

statement([semicolon,endsy]+fsys); {\*解析语句部分\*}

while sy in [semicolon]+statbegsys do

begin

if sy = semicolon

then insymbol

else error(14);

statement([semicolon,endsy]+fsys);

end;

if sy = endsy

then insymbol

else error(57);

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

end { block };

procedure interpret;

var ir : order ; { instruction buffer }

pc : integer; { program counter }

t : integer; { top stack index }

b : integer; { base index }

h1,h2,h3: integer;

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

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

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

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

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

record

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

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

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

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

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

end;

procedure dump;

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 }

procedure inter0;

begin

case ir.f of

0 : begin { load addrss }

t := t + 1;

if t > stacksize

then ps := stkchk

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

end;

1 : begin { load value }

t := t + 1;

if t > stacksize

then ps := stkchk

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

end;

2 : begin { load indirect }

t := t + 1;

if t > stacksize

then ps := stkchk

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

end;

3 : begin { update display }

h1 := ir.y;

h2 := ir.x;

h3 := b;

repeat

display[h1] := h3;

h1 := h1-1;

h3 := s[h3+2].i

until h1 = h2

end;

8 : case ir.y of

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;

if t > stacksize

then ps := stkchk

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

end;

18 : begin

t := t+1;

if t > stacksize

then ps := stkchk

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

end;

end;

9 : s[t].i := s[t].i + ir.y; { offset }

end { case ir.y }

end; { inter0 }

procedure inter1;

var h3, h4: integer;

begin

case ir.f of

10 : pc := ir.y ; { jump }

11 : begin { conditional jump }

if not s[t].b

then pc := ir.y;

t := t - 1

end;

12 : begin { switch }

h1 := s[t].i;

t := t-1;

h2 := ir.y;

h3 := 0;

repeat

if code[h2].f <> 13

then begin

h3 := 1;

ps := caschk

end

else if code[h2].y = h1

then begin

h3 := 1;

pc := code[h2+1].y

end

else h2 := h2 + 2

until h3 <> 0

end;

14 : begin { for1up }

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

if h1 <= s[t].i

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

else begin

t := t - 3;

pc := ir.y

end

end;

15 : begin { for2up }

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

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

if h1 <= s[t].i

then begin

s[h2].i := h1;

pc := ir.y

end

else t := t-3;

end;

16 : begin { for1down }

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

if h1 >= s[t].i

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

else begin

pc := ir.y;

t := t - 3

end

end;

17 : begin { for2down }

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

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

if h1 >= s[t].i

then begin

s[h2].i := h1;

pc := ir.y

end

else t := t-3;

end;

18 : begin { mark stack }

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

if t+h1 > stacksize

then ps := stkchk

else begin

t := t+5;

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

s[t].i := ir.y

end

end;

19 : begin { call }

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

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

h3 := tab[h2].lev;

display[h3+1] := h1;

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

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

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

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

for h3 := t+1 to h4 do

s[h3].i := 0;

b := h1;

t := h4;

pc := tab[h2].adr;

if stackdump

then dump

end;

end { case }

end; { inter1 }

procedure inter2;

begin

case ir.f of

20 : begin { index1 }

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

h2 := atab[h1].low;

h3 := s[t].i;

if h3 < h2

then ps := inxchk

else if h3 > atab[h1].high

then ps := inxchk

else begin

t := t-1;

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

end

end;

21 : begin { index }

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

h2 := atab[h1].low;

h3 := s[t].i;

if h3 < h2

then ps := inxchk

else if h3 > atab[h1].high

then ps := inxchk

else begin

t := t-1;

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

end

end;

22 : begin { load block }

h1 := s[t].i;

t := t-1;

h2 := ir.y+t;

if h2 > stacksize

then ps := stkchk

else while t < h2 do

begin

t := t+1;

s[t] := s[h1];

h1 := h1+1

end

end;

23 : begin { copy block }

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

h2 := s[t].i;

h3 := h1+ir.y;

while h1 < h3 do

begin

s[h1] := s[h2];

h1 := h1+1;

h2 := h2+1

end;

t := t-2

end;

24 : begin { literal }

t := t+1;

if t > stacksize

then ps := stkchk

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

end;

25 : begin { load real }

t := t+1;

if t > stacksize

then ps := stkchk

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

end;

26 : begin { float }

h1 := t-ir.y;

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

end;

27 : begin { read }

if eof(prd)

then ps := redchk

else case ir.y of

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

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

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

end;

t := t-1

end;

28 : begin { write string }

h1 := s[t].i;

h2 := ir.y;

t := t-1;

chrcnt := chrcnt+h1;

if chrcnt > lineleng

then ps := lngchk;

repeat

write(prr,stab[h2]);

h1 := h1-1;

h2 := h2+1

until h1 = 0

end;

29 : begin { write1 }

chrcnt := chrcnt + fld[ir.y];

if chrcnt > lineleng

then ps := lngchk

else case ir.y of

1 : write(prr,s[t].i:fld[1]);

2 : write(prr,s[t].r:fld[2]);

3 : if s[t].b

then write('true')

else write('false');

4 : write(prr,chr(s[t].i));

end;

t := t-1

end;

end { case }

end; { inter2 }

procedure inter3;

begin

case ir.f of

30 : begin { write2 }

chrcnt := chrcnt+s[t].i;

if chrcnt > lineleng

then ps := lngchk

else case ir.y of

1 : write(prr,s[t-1].i:s[t].i);

2 : write(prr,s[t-1].r:s[t].i);

3 : if s[t-1].b

then write('true')

else write('false');

end;

t := t-2

end;

31 : ps := fin;

32 : begin { exit procedure }

t := b-1;

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

b := s[b+3].i

end;

33 : begin { exit function }

t := b;

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

b := s[b+3].i

end;

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

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

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

37 : begin

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

if chrcnt > lineleng

then ps := lngchk

else write(prr,s[t-2].r:s[t-1].i:s[t].i);

t := t-3

end;

38 : begin { store }

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

t := t-2

end;

39 : begin

t := t-1;

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

end;

end { case }

end; { inter3 }

procedure inter4;

begin

case ir.f of

40 : begin

t := t-1;

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

end;

41 : begin

t := t-1;

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

end;

42 : begin

t := t-1;

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

end;

43 : begin

t := t-1;

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

end;

44 : begin

t := t-1;

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

end;

45 : begin

t := t-1;

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

end;

46 : begin

t := t-1;

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

end;

47 : begin

t := t-1;

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

end;

48 : begin

t := t-1;

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

end;

49 : begin

t := t-1;

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

end;

end { case }

end; { inter4 }

procedure inter5;

begin

case ir.f of

50 : begin

t := t-1;

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

end;

51 : begin

t := t-1;

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

end;

52 : begin

t := t-1;

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

end;

53 : begin

t := t-1;

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

end;

54 : begin

t := t-1;

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

end;

55 : begin

t := t-1;

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

end;

56 : begin

t := t-1;

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

end;

57 : begin

t := t-1;

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

end;

58 : begin

t := t-1;

if s[t+1].i = 0

then ps := divchk

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

end;

59 : begin

t := t-1;

if s[t+1].i = 0

then ps := divchk

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

end;

end { case }

end; { inter5 }

procedure inter6;

begin

case ir.f of

60 : begin

t := t-1;

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

end;

61 : begin

t := t-1;

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

end;

62 : if eof(prd)

then ps := redchk

else readln;

63 : begin

writeln(prr);

lncnt := lncnt+1;

chrcnt := 0;

if lncnt > linelimit

then ps := linchk

end

end { case };

end; { inter6 }

begin { interpret }

s[1].i := 0;

s[2].i := 0;

s[3].i := -1;

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

display[0] := 0;

display[1] := 0;

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

b := 0;

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

lncnt := 0;

ocnt := 0;

chrcnt := 0;

ps := run;

fld[1] := 10;

fld[2] := 22;

fld[3] := 10;

fld[4] := 1;

repeat

ir := code[pc];

pc := pc+1;

ocnt := ocnt+1;

case ir.f div 10 of

0 : inter0;

1 : inter1;

2 : inter2;

3 : inter3;

4 : inter4;

5 : inter5;

6 : inter6;

end; { case }

until ps <> run;

if ps <> fin

then begin

writeln(prr);

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

case ps of

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

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

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

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

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

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

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

end;

h1 := b;

blkcnt := 10; { post mortem dump }

repeat

writeln( prr );

blkcnt := blkcnt-1;

if blkcnt = 0

then h1 := 0;

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

if h1 <> 0

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

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

while h2 <> 0 do

with tab[h2] do

begin

if obj = vvariable

then if typ in stantyps

then begin

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

if normal

then h3 := h1+adr

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

case typ of

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

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

bools: if s[h3].b

then writeln(prr,'true')

else writeln(prr,'false');

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

end

end;

h2 := link

end;

h1 := s[h1+3].i

until h1 < 0

end;

writeln(prr);

writeln(prr,ocnt,' steps');

end; { interpret }

procedure setup;

{\*初始化\*}

begin

{\*初始化保留字\*}

key[1] := 'and ';

key[2] := 'array ';

key[3] := 'begin ';

key[4] := 'case ';

key[5] := 'const ';

key[6] := 'div ';

key[7] := 'do ';

key[8] := 'downto ';

key[9] := 'else ';

key[10] := 'end ';

key[11] := 'for ';

key[12] := 'function ';

key[13] := 'if ';

key[14] := 'mod ';

key[15] := 'not ';

key[16] := 'of ';

key[17] := 'or ';

key[18] := 'procedure ';

key[19] := 'program ';

key[20] := 'record ';

key[21] := 'repeat ';

key[22] := 'then ';

key[23] := 'to ';

key[24] := 'type ';

key[25] := 'until ';

key[26] := 'var ';

key[27] := 'while ';

{\*初始化保留字symbol\*}

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;

{\*初始化特殊字符symbol\*}

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;

{\*在符号表中登入标准常量、类型名、函数、过程\*}

begin

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

end;

begin { main }

setup;

constbegsys := [ plus, minus, intcon, realcon, charcon, ident ]; {\*常量头符号集\*}

typebegsys := [ ident, arraysy, recordsy ]; {\*类型头符号集\*}

blockbegsys := [ constsy, typesy, varsy, procsy, funcsy, beginsy ]; {\*块头符号集\*}

facbegsys := [ intcon, realcon, charcon, ident, lparent, notsy ]; {\*因子头符号集\*}

statbegsys := [ beginsy, ifsy, whilesy, repeatsy, forsy, casesy ]; {\*语句头符号集\*}

stantyps := [ notyp, ints, reals, bools, chars ]; {\*标准类型\*}

lc := 0;

ll := 0;

cc := 0;

ch := ' ';

errpos := 0;

errs := [];

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

writeln;

write( 'Source input file ?');

readln( inf );

assign( psin, inf );

reset( psin );

write( 'Source listing file ?');

readln( outf );

assign( psout, outf );

rewrite( psout );{\*psout输出运行过程\*}

assign ( prd, 'con' );

write( 'result file : ' );

readln( fprr );

assign( prr, fprr );{\*prr输出运行结果\*}

reset ( prd );

rewrite( prr );

t := -1;

a := 0;

b := 1;

sx := 0;

c2 := 0;

display[0] := 1;

iflag := false;

oflag := false;

skipflag := false;

prtables := false;

stackdump := false;

insymbol; {\*读入第一个符号\*}

{\*主程序解析部分\*}

if sy <> programsy

then error(3)

else begin

insymbol;

if sy <> ident

then error(2)

else begin

progname := id;

insymbol;

if sy <> lparent

then error(9)

else repeat

insymbol;

if sy <> ident

then error(2)

else begin

if id = 'input '

then iflag := true {\*有输入\*}

else if id = 'output '

then oflag := true {\*有输出\*}

else error(0);

insymbol

end

until sy <> comma;

if sy = rparent

then insymbol

else error(4);

if not oflag then error(20)

end

end;

enterids;

with btab[1] do

begin

last := t;

lastpar := 1;

psize := 0;

vsize := 0;

end;

block( blockbegsys + statbegsys, false, 1 );

if sy <> period

then error(2);

emit(31); { halt }

if prtables

then printtables;

if errs = [] {\*无错误则开始解析，有错误输出错误\*}

then interpret

else begin

writeln( psout );

writeln( psout, 'compiled with errors' );

writeln( psout );

errormsg;

end;

writeln( psout );

close( psout );

close( prr )

end.