program pl0(input,output,fin) ; { version 1.0 oct.1989 }

{ PL/0 compiler with code generation }

const norw = 13; { no. of reserved words } {\*保留字的数目\*}

txmax = 100;{ length of identifier table } {\*符号表长度\*}

nmax = 14; { max. no. of digits in numbers } {\*数字的最大长度\*}

al = 10; { length of identifiers } {\*标识符的最大长度\*}

amax = 2047;{ maximum address } {\*相对地址最大值\*}

levmax = 3; { maximum depth of block nesting } {\*最大嵌套层数\*}

cxmax = 200;{ size of code array } {\*生成目标代码数组最大长度\*}

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 ); {\*symbol的宏定义为一个枚举\*}

alfa = packed array[1..al] of char; {\*alfa宏定义为含有a1个元素的合并数组，为标识符的类型\*}

objecttyp = (constant,variable,prosedure); {\*objecttyp的宏定义为一个枚举\*}

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

fct = ( lit,opr,lod,sto,cal,int,jmp,jpc,red,wrt ); { functions } {\*fct为一个枚举，其实是PCODE的各条指令\*}

instruction = packed record {\*instruction声明为一个记录类型\*}

f : fct; { function code }{\*函数代码\*}

l : 0..levmax; { level } {\*嵌套层次\*}

a : 0..amax; { displacement address }{\*相对位移地址\*}

end;

{ 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

}

{全局变量定义}

var ch : char; { last character read } {\*最后读出的字符\*}

sym: symbol; { last symbol read } {\*最近识别出来符号类型\*}

id : alfa; { last identifier read }{\*最后读出来的识别符\*}

num: integer; { last number read } {\*最后读出来的数字\*}

cc : integer; { character count } {\*行缓冲区指针\*}

ll : integer; { line length } {\*行缓冲区长度\*}

kk,err: integer;

cx : integer; { code allocation index } {\*代码分配指针\*}

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

a : alfa; {\*用来存储symbol的变量\*}

code : array[0..cxmax] of instruction; {\*用来保存编译后的PCODE代码，最大容量为cxmax\*}

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

wsym : array[1..norw] of symbol; {\*保留字表中每个保留字对应的symbol类型\*}

ssym : array[char] of symbol;{\*符号对应的symbol类型\*}

mnemonic : array[fct] of {\*助记符\*}

packed array[1..5] of char;

declbegsys, statbegsys, facbegsys : symset; {\*声明开始，表达式开始、项开始的符号集合\*}

table : array[0..txmax] of {\*定义符号表\*}

record {\*表中的元素类型是记录类型\*}

name : alfa; {\*元素名\*}

case kind: objecttyp of {\*根据符号的类型保存相应的信息\*}

constant : (val:integer );{\*如果是常量，val中保存常量的值\*}

variable,prosedure: (level,adr: integer ) {\*如果是变量或过程，保存存放层数和偏移地址\*}

end;

fin : text; { source program file } {\*源代码文件\*}

sfile: string; { source program file name } {\*源程序文件名\*}

procedure error( n : integer ); {\*错误处理程序，在这里是报错功能，指出错位置及错误性质\*}

begin

writeln( '\*\*\*\*', ' ':cc-1, '^', n:2 ); {\*报错提示信息，'^'指向出错位置，并提示错误类型\*}

err := err+1 {\*错误次数+1\*}

end; { error }

procedure getsym; {\*词法分析程序\*}

var i,j,k : integer;{\*声明计数变量\*}

procedure getch;

begin

if cc = ll {get character to end of line } {\*如果读完了一行（行指针与该行长度相等）\*}

then begin { read next line } {\*开始读取下一行\*}

if eof(fin) {\*如果到达文件末尾\*}

then begin

writeln('program incomplete'); {\*报错\*}

close(fin); {\*关闭文件\*}

exit; {\*退出\*}

end;

ll := 0;{\*将行长度重置\*}

cc := 0;{\*将行指针重置\*}

write(cx:4,' '); { print code address } {\*输出代码地址，宽度为4\*}

while not eoln(fin) do {\*当没有到行末时\*}

begin

ll := ll+1; {\*将行缓冲区的长度+1\*}

read(fin,ch); {\*从文件中读取一个字符到ch中\*}

write(ch); {\*控制台输出ch\*}

line[ll] := ch {\*把这个字符放到当前行末尾\*}

end;

writeln; {\*换行\*}

readln(fin); {\*源文件读取从下一行开始\*}

ll := ll+1; {\*行长度计数加一\*}

line[ll] := ' ' { process end-line } {\*行数组最后一个元素为空格\*}

end;

cc := cc+1; {\*行指针+1\*}

ch := line[cc] {\*读取下一个字符，将字符放进全局变量ch\*}

end; { getch }

begin { procedure getsym; }{\*标识符识别开始\*}

while ch = ' ' do {\*去除空字符\*}

getch; {\*调用上面的getch过程\*}

if ch in ['a'..'z'] {\*如果识别到字母，那么有可能是保留字或标识符\*}

then begin { identifier of reserved word } {\*开始识别\*}

k := 0; {\*标识符指针置零，这个量用来统计标识符长度\*}

repeat {\*循环\*}

if k < al {\*如果k的大小小于标识符的最大长度\*}

then begin

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

a[k] := ch {\*将ch写入标识符暂存变量a\*}

end;

getch{\*获取下一个字符\*}

until not( ch in ['a'..'z','0'..'9']);{\*直到读出的不是数字或字母的时候，标识符结束\*}

if k >= kk { kk : last identifier length} {\*若k比kk大\*}

then kk := k {\*kk记录当前标识符的长度k\*}

else repeat {\*循环\*}

a[kk] := ' '; {\*标识符最后一位为空格\*}

kk := kk-1 {\*k--\*}

until kk = k; {\*直到kk等于当前标识符的长度，这样做的意义是防止上一个标识符存在a中的内容影响到当前标识符，

比如上一个标识符为“qwerty”，现在的标识符为“abcd”，如果不清后几位则a中会保存"abcdty"，这显然是错误的\*}

id := a; {\*id保存标识符名\*}

i := 1; {\*i指向第一个保留字\*}

j := norw; { binary search reserved word table } {\*二分查找保留字表，将j设为保留字的最大数目\*}

repeat

k := (i+j) div 2; {\*再次用到k，但这里只是作为二分查找的中间变量\*}

if id <= word[k] {\*若当前标识符小于或等于保留字表中的第k个，这里的判断依据的是字典序，那么我们可以推测符号表是按照字典序保存的\*}

then j := k-1; {\*j = k-1\*}

if id >= word[k] {\*若当前标识符大于或等于保留字表中的第k个\*}

then i := k+1 {\*i = k+1\*}

until i > j; {\*查找结束条件\*}

if i-1 > j {\*找到了\*}

then sym := wsym[k] {\*将找到的保留字类型赋给sym\*}

else sym := ident {\*未找到则把sym置为ident类型，表示是标识符\*}

end

else if ch in ['0'..'9'] {\*如果字符是数字\*}

then begin { number }

k := 0; {\*这里的k用来记录数字的位数\*}

num := 0; {\*num保存数字\*}

sym := number; {\*将标识符设置为数字\*}

repeat {\*循环开始\*}

num := 10\*num+(ord(ch)-ord('0')); {\*将数字字符转换为数字并拼接起来赋给num\*}

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

getch {\*继续读字符\*}

until not( ch in ['0'..'9']); {\*直到输入的不再是数字\*}

if k > nmax {\*如果数字的位数超过了数字允许的最大长度\*}

then error(30) {\*报错\*}

end

else if ch = ':' {\*当字符不是数字或字母，而是':'时\*}

then begin

getch; {\*读下一个字符\*}

if ch = '=' {\*如果下一个字符是'='\*}

then begin

sym := becomes; {\*将标识符sym设置为becomes，表示复制\*}

getch {\*读下一个字符\*}

end

else sym := nul {\*否则，将标识符设置为nul，表示非法\*}

end

else if ch = '<' {\*当读到的字符是'<'时\*}

then begin

getch; {\*读下一个字符\*}

if ch = '=' {\*若读到的字符是'='\*}

then begin

sym := leq; {\*则sym为leq,表示小于等于\*}

getch {\*读下一个字符\*}

end

else if ch = '>' {\*若读到的字符是'>'\*}

then begin

sym := neq; {\*则sym为neq,表示不等于\*}

getch {\*读下一个字符\*}

end

else sym := lss {\*否则,sym设为lss,表示小于\*}

end

else if ch = '>' {\*若读到的是'>'\*}

then begin

getch; {\*读下一个字符\*}

if ch = '=' {\*若读到的是'='\*}

then begin

sym := geq; {\*sym设为geq,表示大于等于\*}

getch {\*读下一个字符\*}

end

else sym := gtr {\*否则,sym设为gtr,表示大于\*}

end

else begin {\*若非上述几种符号\*}

sym := ssym[ch]; {\*从ssym表中查到此字符对应的类型,赋给sym\*}

getch {\*读下一个字符\*}

end

end; { getsym }

procedure gen( x: fct; y,z : integer ); {\*目标代码生成过程,x表示PCODE指令,y,z是指令的两个操作数\*}

begin

if cx > cxmax {\*如果当前生成代码的行数cx大于允许的最大长度cxmax\*}

then begin

writeln('program too long'); {\*输出报错信息\*}

close(fin); {\*关闭文件\*}

exit {\*退出程序\*}

end;

with code[cx] do {\*如果没有超出,对目标代码cx\*}

begin

f := x; {\*令其f为x\*}

l := y; {\*令其l为y\*}

a := z {\*令其a为z\*}

{\*这三句对应着code身为instruction类型的三个属性\*}

end;

cx := cx+1 {\*将当前代码行数之计数加一\*}

end; { gen }

procedure test( s1,s2 :symset; n: integer ); {\*测试当前字符合法性过程,用于错误语法处理,若不合法则跳过单词值只读到合法单词为止\*}

begin

if not ( sym in s1 ){\*如果当前符号不在s1中\*}

then begin

error(n); {\*报n号错误\*}

s1 := s1+s2; {\*将s1赋值为s1和s2的集合\*}

while not( sym in s1) do {\*这个while的本质是pass掉所有不合法的符号,以恢复语法分析工作\*}

getsym {\*获得下一个标识符\*}

end

end; { test }

procedure block( lev,tx : integer; fsys : symset ); {\*进行语法分析的主程序,lev表示语法分析所在层次,tx是当前符号表指针,fsys是用来恢复错误的单词集合\*}

var dx : integer; { data allocation index } {\*数据地址索引\*}

tx0: integer; { initial table index } {\*符号表初始索引\*}

cx0: integer; { initial code index } {\*初始代码索引\*}

procedure enter( k : objecttyp ); {\*将对象插入到符号表中\*}

begin { enter object into table }

tx := tx+1; {\*符号表序号加一,指向一个空表项\*}

with table[tx] do {\*改变tx序号对应表的内容\*}

begin

name := id; {\*name记录object k的id,从getsym获得\*}

kind := k; {\*kind记录k的类型,为传入参数\*}

case k of {\*根据类型不同会进行不同的操作\*}

constant : begin {\*对常量\*}

if num > amax {\*如果常量的数值大于约定的最大值\*}

then begin

error(30); {\*报30号错误\*}

num := 0 {\*将常量置零\*}

end;

val := num {\*val保存该常量的值,结合上句可以看出,如果超过限制则保存0\*}

end;

variable : begin {\*对变量\*}

level := lev; {\*记录所属层次\*}

adr := dx; {\*记录变量在当前层中的偏移量\*}

dx := dx+1 {\*偏移量+1,位下一次插入做准备\*}

end;

prosedure: level := lev; {\*对过程,记录所属层次\*}

end

end

end; { enter }

function position ( id : alfa ): integer; {\*查找符号表的函数,输入id为需要寻找的符号\*}

var i : integer; {\*声明记录变量\*}

begin

table[0].name := id; {\*把id放到符号表0号位置\*}

i := tx; {\*将i设置为符号表的最后一个位置,因为符号表是栈式结构,因此按层次逆序查找\*}

while table[i].name <> id do {\*如果当前表项的name和id不同\*}

i := i-1; {\*再向前找\*}

position := i {\*&找到了,把位置赋值给position返回\*}

end; { position }

procedure constdeclaration; {\*处理常量声明的过程\*}

begin

if sym = ident {\*如果sym是ident说明是标识符\*}

then begin

getsym; {\*获取下一个sym类型\*}

if sym in [eql,becomes] {\*如果sym是等号或者赋值符号\*}

then begin

if sym = becomes {\*若是赋值符号\*}

then error(1); {\*报一号错误,因为声明应该使用等号\*}

getsym; {\*获取下一个sym类型\*}

if sym = number {\*如果读到的是数字\*}

then begin

enter(constant); {\*将该常量入表\*}

getsym {\*获取下一个sym类型\*}

end

else error(2) {\*如果等号后面不是数字,报2号错误\*}

end

else error(3) {\*如果常量标识符后面接的不是等号或赋值符号,报三号错误\*}

end

else error(4) {\*如果常量声明第一个符号不是标识符,报4号错误\*}

end; { constdeclaration } {\*常量声明结束\*}

procedure vardeclaration;{\*变量声明过程\*}

begin

if sym = ident {\*变量声明要求第一个sym为标识符\*}

then begin

enter(variable);{\*将该变量入表\*}

getsym {\*获取下一个sym类型\*}

end

else error(4) {\*如果第一个sym不是标识符,抛出4号错误\*}

end; { vardeclaration }

procedure listcode; {\*列出PCODE的过程\*}

var i : integer; {\*声明计数变量\*}

begin

for i := cx0 to cx-1 do {\*所有生成的代码\*}

with code[i] do {\*对于每一行代码\*}

writeln( i:4, mnemonic[f]:7,l:3, a:5) {\*格式化输出,分别输出序号,指令的助记符,层次,地址.实际的输出效果和我们实际的PCODE相同\*}

end; { listcode }

procedure statement( fsys : symset ); {\*语句处理的过程\*}

var i,cx1,cx2: integer; {\*定义参数\*}

procedure expression( fsys: symset); {\*处理表达式的过程\*}

var addop : symbol; {\*定义参数\*}

procedure term( fsys : symset); {\*处理项的过程\*}

var mulop: symbol ; {\*定义参数\*}

procedure factor( fsys : symset ); {\*处理因子的处理程序\*}

var i : integer; {\*定义参数\*}

begin

test( facbegsys, fsys, 24 ); {\*测试单词的合法性,判别当前sym是否在facbegsys中,后者在main中定义,如果不在报24号错误\*}

while sym in facbegsys do {\*循环处理因子\*}

begin

if sym = ident {\*如果识别到标识符\*}

then begin

i := position(id); {\*查表,记录其在符号表中的位置,保存至i\*}

if i= 0 {\*如果i为0,表示没查到\*}

then error(11) {\*报11号错误\*}

else

with table[i] do {\*对第i个表项的内容\*}

case kind of {\*按照表项的类型执行不同的操作\*}

constant : gen(lit,0,val); {\*如果是常量类型,生成lit指令,操作数为0,val\*}

variable : gen(lod,lev-level,adr); {\*如果是变量类型,生成lod指令,操作数为lev-level,adr\*}

prosedure: error(21) {\*如果因子处理中识别到了过程标识符,报21号错误\*}

end;

getsym {\*获取下一个sym类型\*}

end

else if sym = number {\*如果识别到数字\*}

then begin

if num > amax {\*判别数字是否超过规定上限\*}

then begin

error(30); {\*超过上限,报30号错误\*}

num := 0 {\*将数字重置为0\*}

end;

gen(lit,0,num); {\*生成lit指令,将num的值放到栈顶\*}

getsym {\*获取下一个sym类型\*}

end

else if sym = lparen {\*如果识别到左括号\*}

then begin

getsym; {\*获取下一个sym类型\*}

expression([rparen]+fsys); {\*调用表达式的过程来处理,递归下降子程序方法\*}

if sym = rparen {\*如果识别到右括号\*}

then getsym {\*获取下一个sym类型\*}

else error(22) {\*报22号错误\*}

end;

test(fsys,[lparen],23) {\*测试结合是否在fsys中,若不是,抛出23号错误\*}

end

end; { factor }

begin { procedure term( fsys : symset);

var mulop: symbol ; } {\*项的分析过程开始\*}

factor( fsys+[times,slash]); {\*项的第一个符号应该是因子,调用因子分析程序\*}

while sym in [times,slash] do {\*如果因子后面是乘/除号\*}

begin

mulop := sym; {\*使用mulop保存当前的运算符\*}

getsym; {\*获取下一个sym类型\*}

factor( fsys+[times,slash] );{\*调用因子分析程序分析运算符后的因子\*}

if mulop = times {\*如果运算符是称号\*}

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

else gen( opr,0,5) {\*生成opr指令,除法指令\*}

end

end; { term }

begin { procedure expression( fsys: symset);

var addop : symbol; } {\*表达式的分析过程开始\*}

if sym in [plus, minus] {\*如果表达式的第一个符号是+/-符号\*}

then begin

addop := sym; {\*保存当前符号\*}

getsym; {\*获取下一个sym类型\*}

term( fsys+[plus,minus]); {\*正负号后面接项,调用项的分析过程\*}

if addop = minus {\*如果符号开头\*}

then gen(opr,0,1) {\*生成opr指令,完成取反运算\*}

end

else term( fsys+[plus,minus]);{\*如果不是符号开头,直接调用项的分析过程\*}

while sym in [plus,minus] do {\*向后面可以接若干个term,使用操作符+-相连,因此此处用while\*}

begin

addop := sym; {\*记录运算符类型\*}

getsym; {\*获取下一个sym类型\*}

term( fsys+[plus,minus] ); {\*调用项的分析过程\*}

if addop = plus {\*如果是加号\*}

then gen( opr,0,2) {\*生成opr指令,完成加法运算\*}

else gen( opr,0,3) {\*否则生成减法指令\*}

end

end; { expression }

procedure condition( fsys : symset ); {\*条件处理过程\*}

var relop : symbol; {\*临时变量\*}

begin

if sym = oddsym {\*如果当天符号是odd运算符\*}

then begin

getsym;{\*获取下一个sym类型\*}

expression(fsys); {\*调用表达式分析过程\*}

gen(opr,0,6) {\*生成opr6号指令,完成奇偶判断运算\*}

end

else begin

expression( [eql,neq,lss,gtr,leq,geq]+fsys); {\*调用表达式分析过程对表达式进行计算\*}

if not( sym in [eql,neq,lss,leq,gtr,geq]) {\*如果存在集合之外的符号\*}

then error(20) {\*报20号错误\*}

else begin

relop := sym; {\*记录当前符号类型\*}

getsym; {\*获取下一个sym类型\*}

expression(fsys);{\*调用表达式分析过程对表达式进行分析\*}

case relop of {\*根据当前符号类型不同完成不同的操作\*}

eql : gen(opr,0,8); {\*如果是等号,生成opr8号指令,判断是否相等\*}

neq : gen(opr,0,9); {\*如果是不等号,生成opr9号指令,判断是否不等\*}

lss : gen(opr,0,10); {\*如果是小于号,生成opr10号指令,判断是否小于\*}

geq : gen(opr,0,11);{\*如果是大于等于号,生成opr11号指令,判断是否大于等于\*}

gtr : gen(opr,0,12); {\*如果是大于号,生成opr12号指令,判断是否大于\*}

leq : gen(opr,0,13);{\*如果是小于等于号,生成opr13号指令,判断是否小于等于\*}

end

end

end

end; { condition }

begin { procedure statement( fsys : symset );

var i,cx1,cx2: integer; } {\*声明处理过程\*}

if sym = ident {\*如果以标识符开始\*}

then begin

i := position(id); {\*i记录该标识符在符号表中的位置\*}

if i= 0 {\*如果返回0则是没找到\*}

then error(11) {\*抛出11号错误\*}

else if table[i].kind <> variable {\*如果在符号表中找到了该符号,但该符号的类型不是变量\*}

then begin { giving value to non-variation } {\*那么现在的操作属于给非变量赋值\*}

error(12); {\*报12号错误\*}

i := 0 {\*将符号表标号置零\*}

end;

getsym; {\*获取下一个sym类型\*}

if sym = becomes {\*如果读到的是赋值符号\*}

then getsym {\*获取下一个sym类型\*}

else error(13); {\*如果读到的不是赋值符号,报13号错误\*}

expression(fsys);{\*赋值符号的后面可以跟表达式,因此调用表达式处理子程序\*}

if i <> 0 {\*如果符号表中找到了合法的符号\*}

then

with table[i] do {\*使用该表项的内容来进行操作\*}

gen(sto,lev-level,adr){\*生成一条sto指令用来将表达式的值写入到相应变量的地址\*}

end

else if sym = callsym {\*如果读到的符号是call关键字\*}

then begin

getsym; {\*获取下一个sym类型\*}

if sym <> ident {\*如果call后面跟的不是标识符\*}

then error(14) {\*报14号错误\*}

else begin {\*如果没有报错\*}

i := position(id);{\*记录当前符号在符号表中的位置\*}

if i = 0 {\*如果没有找到\*}

then error(11) {\*报11号错误\*}

else {\*如果找到了\*}

with table[i] do {\*对第i个表项做如下操作\*}

if kind = prosedure {\*如果该表项的种类为过程\*}

then gen(cal,lev-level,adr) {\*生成cal代码用来实现call操作\*}

else error(15); {\*如果种类不为过程类型,报15号错误\*}

getsym {\*获取下一个sym类型\*}

end

end

else if sym = ifsym {\*如果读到的符号是if关键字\*}

then begin

getsym;{\*获取下一个sym类型\*}

condition([thensym,dosym]+fsys); {\*if后面跟的应该是条件语句,调用条件分析过程\*}

if sym = thensym {\*如果条件语句后面跟的是then关键字的话\*}

then getsym {\*获取下一个sym类型\*}

else error(16);{\*如果条件后面接的不是then,报16号错误\*}

cx1 := cx; {\*记录当前的生成代码位置\*}

gen(jpc,0,0); {\*生成条件跳转指令,跳转位置暂填0\*}

statement(fsys);{\*分析then语句后面的语句\*}

code[cx1].a := cx {\*将之前记录的代码的位移地址改写到现在的生成代码位置(参考instruction类型的结构)\*}

end

else if sym = beginsym {\*如果读到了begin关键字\*}

then begin

getsym;{\*获取下一个sym类型\*}

statement([semicolon,endsym]+fsys); {\*begin后面默认接语句,递归下降分析\*}

while sym in ([semicolon]+statbegsys) do {\*在分析的过程中\*}

begin

if sym = semicolon {\*如果当前的符号是分好\*}

then getsym {\*获取下一个sym类型\*}

else error(10); {\*否则报10号错误\*}

statement([semicolon,endsym]+fsys){\*继续分析\*}

end;

if sym = endsym {\*如果读到了end关键字\*}

then getsym {\*获取下一个sym类型\*}

else error(17) {\*报17号错误\*}

end

else if sym = whilesym {\*如果读到了while关键字\*}

then begin

cx1 := cx; {\*记录当前生成代码的行数指针\*}

getsym;{\*获取下一个sym类型\*}

condition([dosym]+fsys); {\*因为while后需要添加循环条件,因此调用条件语句的分析过程\*}

cx2 := cx;{\*记录在分析完条件之后的生成代码的位置,也是do开始的位置\*}

gen(jpc,0,0); {\*生成一个条件跳转指令,但是跳转位置(a)置零\*}

if sym = dosym {\*条件后应该接do关键字\*}

then getsym {\*获取下一个sym类型\*}

else error(18); {\*如果没接do,报18号错误\*}

statement(fsys); {\*分析处理循环节中的语句\*}

gen(jmp,0,cx1); {\*生成跳转到cx1的地址,既是重新判断一遍当前条件是否满足\*}

code[cx2].a := cx {\*给之前生成的跳转指令设定跳转的位置为当前位置\*}

end

else if sym = readsym {\*如果读到的符号是read关键字\*}

then begin

getsym; {\*获取下一个sym类型\*}

if sym = lparen {\*read的后面应该接左括号\*}

then

repeat {\*循环开始\*}

getsym;{\*获取下一个sym类型\*}

if sym = ident {\*如果第一个sym标识符\*}

then begin

i := position(id);{\*记录当前符号在符号表中的位置\*}

if i = 0{\*如果i为0,说明符号表中没有找到id对应的符号\*}

then error(11){\*报11号错误\*}

else if table[i].kind <> variable {\*如果找到了,但该符号的类型不是变量\*}

then begin

error(12);{\*报12号错误,不能像常量和过程赋值\*}

i := 0 {\*将i置零\*}

end

else with table[i] do {\*如果是变量类型\*}

gen(red,lev-level,adr) {\*生成一条red指令,读取数据\*}

end

else error(4);{\*如果左括号后面跟的不是标识符,报4号错误\*}

getsym;{\*获取下一个sym类型\*}

until sym <> comma{\*知道现在的符号不是都好,循环结束\*}

else error(40); {\*如果read后面跟的不是左括号,报40号错误\*}

if sym <> rparen{\*如果上述内容之后接的不是右括号\*}

then error(22); {\*报22号错误\*}

getsym {\*获取下一个sym类型\*}

end

else if sym = writesym {\*如果读到的符号是write关键字\*}

then begin

getsym; {\*获取下一个sym类型\*}

if sym = lparen {\*默认write右边应该加一个左括号\*}

then begin

repeat{\*循环开始\*}

getsym;{\*获取下一个sym类型\*}

expression([rparen,comma]+fsys);{\*分析括号中的表达式\*}

gen(wrt,0,0);{\*生成一个wrt海曙，用来输出内容\*}

until sym <> comma;{\*知道读取到的sym不是逗号\*}

if sym <> rparen{\*如果内容结束没有右括号\*}

then error(22);{\*报22号错误\*}

getsym{\*获取下一个sym类型\*}

end

else error(40){\*如果write后面没有跟左括号\*}

end;

test(fsys,[],19){\*测试当前字符是否合法,如果没有出现在fsys中,报19号错\*}

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 \*/ }{\*分程序处理过程开始\*}

dx := 3; {\*记录运行栈空间的栈顶位置,设置为3是因为需要预留SL,DL,RA的空间\*}

tx0 := tx;{\*记录当前符号表的栈顶位置\*}

table[tx].adr := cx;{\*符号表当前位置的偏移地址记录下一条生成代码开始的位置\*}

gen(jmp,0,0); { jump from declaration part to statement part }{\*产生一条jmp类型的无条件跳转指令,跳转位置未知\*}

if lev > levmax{\*当前过程所处的层次大于允许的最大嵌套层次\*}

then error(32);{\*报32号错误\*}

repeat{\*循环开始\*}

if sym = constsym {\*如果符号类型是const保留字\*}

then begin

getsym;{\*获取下一个sym类型\*}

repeat{\*循环开始\*}

constdeclaration; {\*处理常量声明\*}

while sym = comma do{\*如果声明常量后接的是逗号,说明常量声明没有结束,进入下一循环\*}

begin

getsym;{\*获取下一个sym类型\*}

constdeclaration {\*处理常量声明\*}

end;

if sym = semicolon{\*如果读到了分号,说明常量声明已经结束了\*}

then getsym{\*获取下一个sym类型\*}

else error(5){\*如果没有分号,报5号错误\*}

until sym <> ident{\*循环直到遇到下一个标志符\*}

end;

if sym = varsym {\*如果读到的是var保留字\*}

then begin

getsym; {\*获取下一个sym类型\*}

repeat {\*循环开始\*}

vardeclaration;{\*处理变量声明\*}

while sym = comma do {\*如果读到了逗号,说明声明未结束,进入循环\*}

begin

getsym;{\*获取下一个sym类型\*}

vardeclaration {\*处理变量声明\*}

end;

if sym = semicolon{\*如果读到了分号,说明所有声明已经结束\*}

then getsym {\*获取下一个sym类型\*}

else error(5) {\*如果未读到分号,则报5号错误\*}

until sym <> ident;{\*循环直到读到下一个标识符为止\*}

end;

while sym = procsym do {\*如果读到proc关键字\*}

begin

getsym; {\*获取下一个sym类型\*}

if sym = ident{\*第一个符号应该是标识符类型\*}

then begin

enter(prosedure);{\*将该符号录入符号表,类型为过程,因为跟在proc后面的一定是过程名\*}

getsym {\*获取下一个sym类型\*}

end

else error(4); {\*如果第一个符号不是标识符类型,报4号错误\*}

if sym = semicolon {\*如果读到了分号,说明proc声明结束\*}

then getsym {\*获取下一个sym类型\*}

else error(5);{\*如果声明过程之后没有跟分号,报5号错误\*}

block(lev+1,tx,[semicolon]+fsys);{\*执行分程序的分析过程\*}

if sym = semicolon{\*递归调用返回后应该接分号\*}

then begin {\*如果接的是分号\*}

getsym;{\*获取下一个sym类型\*}

test( statbegsys+[ident,procsym],fsys,6) {\*测试当前的sym是否合法\*}

end

else error(5) {\*如果接的不是分号,报5号错误\*}

end;

test( statbegsys+[ident],declbegsys,7){\*测试当前的sym是否合法\*}

until not ( sym in declbegsys ); {\*一直循环到sym不在声明符号集中为止\*}

code[table[tx0].adr].a := cx; { back enter statement code's start adr. } {\*将之前生成无条件跳转指令的目标地址指向当前位置\*}

with table[tx0] do {\*对符号表新加记录\*}

begin

adr := cx; { code's start address } {\*记录当前代码的分配为止\*}

end;

cx0 := cx; {\*记录当前代码分配的地址\*}

gen(int,0,dx); { topstack point to operation area } {\*生成int指令,分配dx个空间\*}

statement( [semicolon,endsym]+fsys); {\*调用语法分析程序\*}

gen(opr,0,0); { return } {\*生成0号gen程序,完成返回操作\*}

test( fsys, [],8 ); {\*测试当前状态是否合法,有问题报8号错误\*}

listcode; {\*列出该block所生成的PCODE\*}

end { block };

procedure interpret; {\*解释执行程序\*}

const stacksize = 500;{\*设置栈大小为常量500\*}

var p,b,t: integer; { program-,base-,topstack-register }{\*设置三个寄存器,分别记录下一条指令,基址地址和栈顶指针\*}

i : instruction;{ instruction register } {\*指令寄存器,类型为instruction,显然是为了存放当前指令\*}

s : array[1..stacksize] of integer; { data store }{\*数据栈,大小为stacksize=500个integer}\*

function base( l : integer ): integer;{\*声明计算基地址的函数\*}

var b1 : integer;{\*声明计数变量\*}

begin { find base l levels down } {\*目标是找到相对于现在层次之差为l的层次基址\*}

b1 := b;{\*记录当前层的基地址\*}

while l > 0 do {\*如果层数大于0,即寻找的不是本层\*}

begin

b1 := s[b1];{\*记录当前层数据基址的内容\*}

l := l-1{\*层数--\*}

end;

base := b1 {\*将找到的基地址保存起来\*}

end; { base }

begin

writeln( 'START PL/0' );{\*输出程序开始运行的提示语句\*}

t := 0; {\*将栈顶指针置零\*}

b := 1; {\*将基址地址置为1\*}

p := 0; {\*将指令寄存器置零\*}

s[1] := 0;{\*将数据栈的第一层置零,对应SL\*}

s[2] := 0;{\*将数据栈的第二层置零,对应DL\*}

s[3] := 0;{\*将数据栈的第三层置零,对应RA\*}

repeat{\*循环开始\*}

i := code[p];{\*获取当前需要执行的代码\*}

p := p+1;{\*将指令寄存器+1,以指向下一条置零\*}

with i do {\*针对当前指令\*}

case f of {\*不同类型的指令执行不同操作\*}

lit : begin {\*对lit类型\*}

t := t+1; {\*栈顶指针加1\*}

s[t]:= a; {\*将a操作数的值放入栈顶\*}

end;

opr : case a of { operator } {\*针对opr类型的指令\*}

0 : begin { return } {\*0对应return操作\*}

t := b-1;{\*t取到该层数据栈SL-1的位置,意味着将该层的数据栈全部清空(因为要返回了嘛)\*}

p := s[t+3]; {\*将指令指针指向RA的值,即获得return address\*}

b := s[t+2]; {\*将基址指针指向DL的值,即获得了return之后的基址,因为被调用层次的DL指向调用层次的基址\*}

end;

1 : s[t] := -s[t];{\*1对应取反操作\*}

2 : begin {\*2对应求和操作\*}

t := t-1; {\*栈顶指针退一格\*}

s[t] := s[t]+s[t+1] {\*将栈顶和次栈顶中的数值求和放入新的栈顶,注意运算后的栈顶是下降一格的,下面的运算亦如此\*}

end;

3 : begin {\*3对应做差操作\*}

t := t-1; {\*栈顶指针退格\*}

s[t] := s[t]-s[t+1] {\*次栈顶减栈顶,结果放入新的栈顶\*}

end;

4 : begin {\*4对应乘积操作\*}

t := t-1; {\*栈顶退格\*}

s[t] := s[t]\*s[t+1] {\*栈顶和次栈顶相乘,结果放入新的栈顶\*}

end;

5 : begin {\*5对应相除\*}

t := t-1; {\*栈顶退格\*}

s[t] := s[t]div s[t+1] {\*次栈顶除以栈顶,结果放入新的栈顶\*}

end;

6 : s[t] := ord(odd(s[t])); {\*6对应判断是否栈顶数值为奇数\*}

8 : begin {\*8号对应等值判断\*}

t := t-1; {\*栈顶退格\*}

s[t] := ord(s[t]=s[t+1]) {\*如果栈顶和次栈顶数值相同,栈顶置一,否则置零\*}

end;

9 : begin {\*9号对应不等判断\*}

t := t-1; {\*栈顶退格\*}

s[t] := ord(s[t]<>s[t+1]) {\*如果栈顶和次栈顶数值不同,栈顶置一,否则置零\*}

end;

10: begin {\*10号对应小于判断\*}

t := t-1; {\*栈顶退格\*}

s[t] := ord(s[t]< s[t+1]) {\*如果次栈顶的数值小于栈顶的数值,栈顶置一,否则置零\*}

end;

11: begin {\*11号对应大于等于判断\*}

t := t-1; {\*栈顶退格\*}

s[t] := ord(s[t] >= s[t+1]) {\*如果次栈顶的数值大于等于栈顶的数值,栈顶置一,否则置零\*}

end;

12: begin {\*12号对应着大于判断\*}

t := t-1; {\*栈顶退格\*}

s[t] := ord(s[t] > s[t+1]) {\*如果次栈顶的数值大于栈顶的数值,栈顶置一,否则置零\*}

end;

13: begin {\*13号对应着小于等于判断\*}

t := t-1; {\*栈顶退格\*}

s[t] := ord(s[t] <= s[t+1]) {\*如果次栈顶的数值小于等于栈顶的数值,栈顶置一,否则置零\*}

end;

end;

lod : begin {\*如果是lod指令\*}

t := t+1; {\*栈顶指针指向新栈\*}

s[t] := s[base(l)+a] {\*将与当前数据层层次差为l,层内偏移为a的栈中的数据存到栈顶\*}

end;

sto : begin {\*对于sto指令\*}

s[base(l)+a] := s[t]; { writeln(s[t]); } {\*将当前栈顶的数据保存到与当前层层差为l,层内偏移为a的数据栈中\*}

t := t-1 {\*栈顶退栈\*}

end;

cal : begin { generate new block mark } {\*对于指令\*}

s[t+1] := base(l); {\*由于要生成新的block,因此栈顶压入SL的值\*}

s[t+2] := b; {\*在SL之上压入当前数据区的基址,作为DL\*}

s[t+3] := p; {\*在DL之上压入指令指针,即是指令的断点,作为R\*A}

b := t+1; {\*把当前的数据区基址指向新的SL\*}

p := a; {\*从a的位置继续执行程序,a来自instruction结构体\*}

end;

int : t := t+a; {\*对int指令,将栈顶指针上移a个位置\*}

jmp : p := a; {\*对jmp指令,将指令指针指向a\*}

jpc : begin {\*对于jpc指令\*}

if s[t] = 0 {\*如果栈顶数据为零\*}

then p := a; {\*则将指令指针指向a\*}

t := t-1; {\*栈顶向下移动\*}

end;

red : begin {\*对red指令\*}

writeln('??:'); {\*输出提示信息\*}

readln(s[base(l)+a]); {\*读一行数据,读入到相差l层,层内偏移为a的数据栈中的数据的信息\*}

end;

wrt : begin {\*对wrt指令\*}

writeln(s[t]); {\*输出栈顶的信息\*}

t := t+1 {\*栈顶上移\*}

end

end { with,case }

until p = 0; {\*直到当前指令的指针为0,这意味着主程序返回了,即整个程序已经结束运行了\*}

writeln('END PL/0'); {\*PL/0执行结束\*}

end; { interpret }

begin { main } { \*主函数 \*}

writeln('please input source program file name : '); {\*提示信息,要求用户输入源码的地址\*}

readln(sfile); {\*读入一行保存至sfile\*}

assign(fin,sfile); {\*将文件名字符串变量str付给文件变量fin\*}

reset(fin); {\*打开fin\*}

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

ssym[ch] := nul; {\*将从'A'到';'的符号的ssym都设置为nul,表示不合法\*}

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 '; {\*填写保留字表,注意这里所有字符都预留的相同的长度\*}

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; {\*填写保留字对应的标识符sym的值\*}

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

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

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

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

ssym['.'] := period;

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

ssym[';'] := semicolon; {\*填写对应符号对应的标识符sym的值\*}

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 '; {\*填写助记符表,与PCODE指令一一对应\*}

declbegsys := [ constsym, varsym, procsym ]; {\*表达式开始的符号集合\*}

statbegsys := [ beginsym, callsym, ifsym, whilesym]; {\*语句开始的符号集合\*}

facbegsys := [ ident, number, lparen ]; {\*项开始的符号集合\*}

err := 0; {\*将出错的标识符置零\*}

cc := 0; {\*行缓冲指针置零\*}

cx := 0; {\*成代码行数计数置零\*}

ll := 0; {\*词法分析行缓冲区长度置零\*}

ch := ' '; {\*当前字符设为' '\*}

kk := al; {\*kk的值初始化为0\*}

getsym; {\*获取第一个词的标识符\*}

block( 0,0,[period]+declbegsys+statbegsys ); {\*执行主程序block\*}

if sym <> period {\*如果符号不是句号\*}

then error(9); {\*报⑨号错误\*}

if err = 0 {\*如果err为0表示没有错误\*}

then interpret {\*始解释执行生成的PCODE代码}\*

else write('ERRORS IN PL/0 PROGRAM'); {\*否则出现了错误,报错\*}

writeln; {\*换行\*}

close(fin); {\*关闭源文件程序\*}

readln(sfile); {\*读取PL/0源程序\*}

end.