编译原理实验报告

陈铭涛

16340024

### 实验内容

* 1. 在计算机上实现PL0语言的编译程序;
  2. 扩展PL0语言的功能,并在计算机上实现.

### 第一部分要求与报告

* 1. 找到PASCAL编译系统( Delphi系统也可以);

在本次实验中，在 Unix 系统下安装了 Free Pascal Compiler version 3.0.4进行实验。

* 1. 在PASCAL系统上运行PL0编译程序,需要对PL0编译程序作一些修改﹑调试;

在实验中对给出的 PL0程序进行了一定的修改，由于其中有不少语法已经不再被Free Pascal编译器所支持，所以需要对其实现做出调整。

首先，对给出代码中的中文字符进行修正，主要是将中文破折号改为减号，中文引号改为英文引号，同时对将≥，≤以及≠作为判断符号的情况进行了修正，由于符号表以字符为键，这里选择了使用~代表不等于，$代表小于等于，^代表大于等于。然后将程序中使用 Pascal 保留字作为变量名的情况进行修正,将其改为缩短的简写。另外，由于 Free Pascal 编译器不支持跨procedure进行跳转，因此将程序出现错误后的跳转改为了使用 exit 进行退出。

* 1. 在PASCAL系统中,为PL0的编译程序建立输入文件和输出文件;

此处进行的修改是在程序启动时要求用户输入 PL0源程序文件名，然后在每次 getch 调用中从文件读取字符。程序运行将会输出两个文件： intermediate.txt为生成的中间代码，result.txt为中间代码解释运行过程中产生的栈的数据。

### 第二部分要求与报告

* 1. 在PL0语言中增加Read和Write语句;

对于第一步时使用的PL0源程序代码，将原先固定的数值赋值修改为了使用 read 语句从键盘中读取数值。并在调用子程序后使用 write 语句将计算结果输出至命令行中。

* 1. 修改PL0编译程序, 使得PL0源程序可以使用Read和Write语句, 从文件(或键盘)输入数据,并可以向文件(或屏幕)写数据.

修改的主要步骤为字典和符号表中添加 read 和 write的字符串，使得getsym获取符号时可以读取这两个字符串，其中在word表中添加时要添加至正确位置，因为getsym 查找时使用的是二分查找法。然后，在助记符表中添加read和 write 对应的助记符RED 和 WRT，使其反映在输出的中间代码中。接下来，在statement 中添加读取 read 和 write 参数的过程，read 中的参数使用逗号分隔，将参数变量的地址写入中间代码中。最后，改写interpret 子程序，添加对 read 和 write 的处理，read 中调用 Pascal 的 read 函数来从键盘中读取输入的数值，write 则调用writeln函数将输出的值打印为屏幕上的一行。

编译程序的其他部分与第一部分的程序一样，使用方法同样为从键盘中读取输入源文件文件名，并将中间代码和栈数据输出。

### 第一部分的代码与数据

PL0编译程序（PL0.pas）

program PL0;

{带有代码生成的PL0编译程序}

label 99;

{fpc 编译器不允许跨procedure 的 goto，因此此处 label 没有作用}

const

norw = 11; {保留字的个数}

txmax = 100; {标识符表长度}

nmax = 14; {数字的最大位数}

al = 10; {标识符的长度}

amax = 2047; {最大地址}

levmax = 3; {程序体嵌套的最大深度}

cxmax = 200; {代码数组的大小}

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 );

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

obj = (constant, variable, prcd);

symset = set of symbol;

fct = (lit, opr, lod, sto, cal, int, jmp, jpc); {functions}

instruction = packed record

f : fct; {功能码}

l : 0..levmax; {相对层数}

a : 0..amax; {相对地址}

end;

{LIT 0,a : 取常数a

OPR 0,a : 执行运算a

LOD l,a : 取层差为l的层﹑相对地址为a的变量

STO l,a : 存到层差为l的层﹑相对地址为a的变量

CAL l,a : 调用层差为l的过程

INT 0,a : t寄存器增加a

JMP 0,a : 转移到指令地址a处

JPC 0,a : 条件转移到指令地址a处 }

var

ch : char; {最近读到的字符}

sym : symbol; {最近读到的符号}

id : alfa; {最近读到的标识符}

num : integer; {最近读到的数}

cc : integer; {当前行的字符计数}

ll : integer; {当前行的长度}

kk, err : integer;

cx : integer; {代码数组的当前下标}

line : array [1..81] of char;

a : alfa;

code : array [0..cxmax] of instruction;

word : array [1..norw] of alfa;

wsym : array [1..norw] of symbol;

ssym : array [char] of symbol;

mnemonic : array [fct] of packed array [1..5] of char;

declbegsys, statbegsys, facbegsys : symset;

srcfilename, itmdfilename, resfilename: string;

fin, fitmd, fres: text;

table : array [0..txmax] of

record

name : alfa;

case kind : obj of

constant : (val : integer);

variable, prcd : (level, adr : integer)

end;

procedure error (n : integer);

begin

writeln('\*\*\*\*', ' ' : cc-1, '↑', n : 2); err := err + 1

end {error};

procedure getsym;

var i, j, k : integer;

procedure getch ;

begin

if cc = ll then

begin

if eof(fin) then

begin

write('PROGRAM INCOMPLETE');

close(fin);

close(fitmd);

close(fres);

exit;

end;

ll := 0; cc := 0;

while not eoln(fin) do

begin

ll := ll + 1;

read(fin, ch);

line[ll] := ch;

end;

readln(fin);

ll := ll + 1;

line[ll] := ' ';

end;

cc := cc + 1; ch := line[cc]

end {getch};

begin {getsym}

while ch = ' ' do getch;

if ch in ['a'..'z'] then

begin {标识符或保留字}

k := 0;

repeat

if k < al then

begin

k:= k + 1;

a[k] := ch

end;

getch

until not (ch in ['a'..'z', '0'..'9']);

if k >= kk then kk := k else

repeat a[kk] := ' ';

kk := kk-1

until kk = k;

id := a; i := 1; j := norw;

repeat k := (i+j) div 2;

if id <= word[k] then j := k-1;

if id >= word[k] then i := k + 1

until i > j;

if i-1 > j then begin

sym := wsym[k];

end

else begin

sym := ident;

end;

end

else

if ch in ['0'..'9'] then

begin {数字}

k := 0;

num := 0;

sym := number;

repeat

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

k := k + 1;

getch;

until not (ch in ['0'..'9']);

if k > nmax then

error(30)

end

else

if ch = ':' then

begin

getch;

if ch = '=' then

begin

sym := becomes;

getch;

end

else

sym := nul;

end else

begin

sym := ssym[ch];

getch;

end

end {getsym};

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

begin

if cx > cxmax then

begin

write('PROGRAM TOO LONG');

close(fin);

close(fitmd);

close(fres);

exit;

end;

with code[cx] do

begin f := x; l := y; a := z

end;

cx := cx + 1

end {gen};

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

begin

if not (sym in s1) then

begin

error(n);

s1 := s1 + s2;

while not (sym in s1) do

getsym;

end

end {test};

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

var

dx : integer; {本过程数据空间分配下标}

tx0 : integer; {本过程标识表起始下标}

cx0 : integer; {本过程代码起始下标}

procedure enter(k : obj);

begin {把obj填入符号表中}

tx := tx +1;

with table[tx] do

begin

name := id;

kind := k;

case k of

constant : begin

if num > amax then

begin

error(30);

num := 0

end;

val := num

end;

variable : begin

level := lev;

adr := dx;

dx := dx +1;

end;

prcd : level := lev

end

end

end {enter};

function position(id : alfa) : integer;

var i : integer;

begin {在标识符表中查标识符id}

table[0].name := id; i := tx;

while table[i].name <> id do

i := i-1;

position := i

end {position};

procedure constdeclaration;

begin

if sym = ident then

begin getsym;

if sym in [eql, becomes] then

begin

if sym = becomes then error(1);

getsym;

if sym = number then

begin

enter(constant);

getsym;

end

else error(2);

end else error(3);

end else error(4);

end {constdeclaration};

procedure vardeclaration;

begin

if sym = ident then

begin enter(variable); getsym

end else error(4)

end {vardeclaration};

procedure listcode;

var i : integer;

begin {列出本程序体生成的代码}

For i := cx0 To cx-1 Do

With code[i] Do

writeln(fitmd, i, mnemonic[f] : 5, l : 3, a : 5)

end {listcode};

procedure statement(fsys : symset);

var i, cx1, cx2 : integer;

procedure expression(fsys : symset);

var addop : symbol;

procedure term(fsys : symset);

var mulop : symbol;

procedure factor(fsys : symset);

var i : integer;

begin

test(facbegsys, fsys, 24);

while sym in facbegsys do

begin

if sym = ident then

begin

i := position(id);

if i = 0 then error(11) else

with table[i] do

case kind of

constant : gen(lit, 0, val);

variable : gen(lod, lev-level, adr);

prcd : error(21)

end;

getsym;

end else

if sym = number then

begin

if num > amax then

begin

error(30);

num := 0

end;

gen(lit, 0, num);

getsym;

end

else

if sym = lparen then

begin

getsym;

expression([rparen]+fsys);

if sym = rparen then getsym

else error(22)

end;

test(fsys, [lparen], 23)

end

end {factor};

begin {term}

factor(fsys+[times, slash]);

while sym in [times, slash] do

begin

mulop := sym;

getsym;

factor(fsys+[times, slash]);

if mulop = times then gen(opr, 0, 4)

else gen(opr, 0, 5)

end

end {term};

begin {expression}

if sym in [plus, minus] then

begin

addop := sym;

getsym;

term(fsys+[plus, minus]);

if addop = minus then gen(opr, 0, 1)

end

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

while sym in [plus, minus] do

begin

addop := sym;

getsym;

term(fsys+[plus, minus]);

if addop = plus then gen(opr, 0, 2)

else gen(opr, 0, 3)

end

end {expression};

procedure condition(fsys : symset);

var relop : symbol;

begin

if sym = oddsym then

begin

getsym;

expression(fsys);

gen(opr, 0, 6)

end else

begin

expression([eql, neq, lss, gtr, leq, geq] + fsys);

if not (sym in [eql, neq, lss, leq, gtr, geq]) then

error(20) else

begin

relop := sym;

getsym;

expression(fsys);

case relop of

eql : gen(opr, 0, 8);

neq : gen(opr, 0, 9);

lss : gen(opr, 0, 10);

geq : gen(opr, 0, 11);

gtr : gen(opr, 0, 12);

leq : gen(opr, 0, 13);

end

end

end

end {condition};

begin {statement}

if sym = ident then

begin i := position(id);

if i = 0 then error(11) else

if table[i].kind <> variable then

begin {对非变量赋值}

error(12);

i := 0;

end;

getsym;

if sym = becomes then

getsym

else error(13);

expression(fsys);

if i <> 0 then

with table[i] do gen(sto, lev-level, adr);

end else

if sym = callsym then

begin getsym;

if sym <> ident then error(14) else

begin

i := position(id);

if i = 0 then error(11) else

with table[i] do

if kind = prcd then

gen(cal, lev-level, adr)

else error(15);

getsym;

end

end else

if sym = ifsym then

begin

getsym;

condition([thensym, dosym]+fsys);

if sym = thensym then

getsym

else error(16);

cx1 := cx;

gen(jpc, 0, 0);

statement(fsys);

code[cx1].a := cx

end else

if sym = beginsym then

begin

getsym;

statement([semicolon, endsym]+fsys);

while sym in [semicolon]+statbegsys do

begin

if sym = semicolon then

getsym

else error(10);

statement([semicolon, endsym]+fsys)

end;

if sym = endsym then

getsym

else error(17)

end else

if sym = whilesym then

begin

cx1 := cx;

getsym;

condition([dosym]+fsys);

cx2 := cx;

gen(jpc, 0, 0);

if sym = dosym then

getsym

else error(18);

statement(fsys);

gen(jmp, 0, cx1);

code[cx2].a := cx

end;

test(fsys, [ ], 19);

end {statement};

begin {block}

dx := 3;

tx0 := tx;

table[tx].adr := cx;

gen(jmp, 0, 0);

if lev > levmax then error(32);

repeat

if sym = constsym then

begin getsym;

repeat

constdeclaration;

while sym = comma do

begin

getsym;

constdeclaration;

end;

if sym = semicolon then

getsym

else error(5)

until sym <> ident

end;

if sym = varsym then

begin

getsym;

repeat

vardeclaration;

while sym = comma do

begin

getsym;

vardeclaration;

end;

if sym = semicolon then

getsym

else error(5);

until sym <> ident;

end;

while sym = procsym do

begin getsym;

if sym = ident then

begin

enter(prcd);

getsym;

end

else error(4);

if sym = semicolon then

getsym

else error(5);

block(lev+1, tx, [semicolon]+fsys);

if sym = semicolon then

begin

getsym;

test(statbegsys+[ident, procsym], fsys, 6)

end

else error(5)

end;

test(statbegsys+[ident], declbegsys, 7)

until not (sym in declbegsys);

code[table[tx0].adr].a := cx;

with table[tx0] do

begin

adr := cx; {代码开始地址}

end;

cx0 := cx;

gen(int, 0, dx);

statement([semicolon, endsym]+fsys);

gen(opr, 0, 0); {生成返回指令}

test(fsys, [ ], 8);

listcode;

end {block};

procedure interpret;

const stacksize = 500;

var p, b, t : integer; {程序地址寄存器, 基地址寄存器,栈顶地址寄存器}

i : instruction; {指令寄存器}

s : array [1..stacksize] of integer; {数据存储栈}

function base(l : integer) : integer;

var b1 : integer;

begin

b1 := b; {顺静态链求层差为l的层的基地址}

while l > 0 do

begin

b1 := s[b1];

l := l-1

end;

base := b1

end {base};

begin

writeln('START PL/0');

t := 0; b := 1; p := 0;

s[1] := 0; s[2] := 0; s[3] := 0;

repeat

i := code[p];

p := p+1;

with i do

case f of

lit : begin

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

end;

opr : case a of {运算}

0 : begin {返回}

t := b-1; p := s[t+3]; b := s[t+2];

end;

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

2 : begin

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

end;

3 : begin

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

end;

4 : begin

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

end;

5 : begin

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

end;

6 : s[t] := ord(odd(s[t]));

8 : begin t := t-1;

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

end;

9: begin t := t-1;

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

end;

10 : begin t := t-1;

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

end;

11: begin t := t-1;

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

end;

12 : begin t := t-1;

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

end;

13 : begin t := t-1;

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

end;

end;

lod : begin

t := t + 1; s[t] := s[base(l) + a]

end;

sto : begin

s[base(l) + a] := s[t];

writeln(fres, s[t]);

t := t-1

end;

cal : begin {generate new block mark}

s[t+1] := base( l ); s[t+2] := b;

s[t+3] := p;

b := t+1; p := a

end;

int : t := t + a;

jmp : p := a;

jpc : begin

if s[t] = 0 then p := a;

t := t-1

end

end {with, case}

until p = 0;

write('END PL/0');

end {interpret};

begin {主程序}

for ch := 'A' to ';' do ssym[ch] := 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] := 'then ';

word[10] := 'var ';

word[11] := 'while ';

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] := thensym; wsym[10] := varsym;

wsym[11] := whilesym;

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

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

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

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

ssym['.'] := period; ssym['~'] := neq;

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

ssym['$'] := leq; ssym['^'] := geq;

ssym[';'] := semicolon;

mnemonic[lit] := 'LIT ';

mnemonic[opr] := 'OPR ';

mnemonic[lod] := 'LOD ';

mnemonic[sto] := 'STO ';

mnemonic[cal] := 'CAL ';

mnemonic[int] := 'INT ';

mnemonic[jmp] := 'JMP ';

mnemonic[jpc] := 'JPC ';

declbegsys := [constsym, varsym, procsym];

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

facbegsys := [ident, number, lparen];

write('Source File: ');

readln(srcfilename);

itmdfilename := 'intermediate.txt';

resfilename := 'result.txt';

assign(fin, srcfilename);

assign(fitmd, itmdfilename);

assign(fres, resfilename);

rewrite(fitmd);

rewrite(fres);

reset(fin);

err := 0;

cc := 0; cx := 0; ll := 0; ch := ' '; kk := al;

getsym;

block(0, 0, [period]+declbegsys+statbegsys);

if sym <> period then error(9);

if err = 0 then interpret

else write('ERRORS IN PL/0 PROGRAM');

close(fin);

close(fitmd);

close(fres);

99 : writeln

end.

PL0源程序代码（source.pl0）

const m = 7, n = 85;

var x, y, z, q, r;

procedure multiply;

var a, b;

begin a := x; b := y; z := 0;

while b > 0 do

begin

if odd b then z := z + a;

a := 2\*a ; b := b/2 ;

end

end;

procedure divide;

var w;

begin r := x; q := 0; w := y;

while w $ r do w := 2\*w ;

while w > y do

begin q := 2\*q; w := w/2;

if w $ r then

begin r := r-w; q := q+1 end

end

end;

procedure gcd;

var f, g ;

begin f := x; g := y;

while f ~ g do

begin

if f < g then g := g-f;

if g < f then f := f-g;

end;

z := f

end;

begin

x := m; y := n; call multiply;

x := 25; y:= 3; call divide;

x := 84; y := 36; call gcd;

end.

输出的中间代码（intermediate.txt）

2INT 0 5

3LOD 1 3

4STO 0 3

5LOD 1 4

6STO 0 4

7LIT 0 0

8STO 1 5

9LOD 0 4

10LIT 0 0

11OPR 0 12

12JPC 0 29

13LOD 0 4

14OPR 0 6

15JPC 0 20

16LOD 1 5

17LOD 0 3

18OPR 0 2

19STO 1 5

20LIT 0 2

21LOD 0 3

22OPR 0 4

23STO 0 3

24LOD 0 4

25LIT 0 2

26OPR 0 5

27STO 0 4

28JMP 0 9

29OPR 0 0

31INT 0 4

32LOD 1 3

33STO 1 7

34LIT 0 0

35STO 1 6

36LOD 1 4

37STO 0 3

38LOD 0 3

39LOD 1 7

40OPR 0 13

41JPC 0 47

42LIT 0 2

43LOD 0 3

44OPR 0 4

45STO 0 3

46JMP 0 38

47LOD 0 3

48LOD 1 4

49OPR 0 12

50JPC 0 72

51LIT 0 2

52LOD 1 6

53OPR 0 4

54STO 1 6

55LOD 0 3

56LIT 0 2

57OPR 0 5

58STO 0 3

59LOD 0 3

60LOD 1 7

61OPR 0 13

62JPC 0 71

63LOD 1 7

64LOD 0 3

65OPR 0 3

66STO 1 7

67LOD 1 6

68LIT 0 1

69OPR 0 2

70STO 1 6

71JMP 0 47

72OPR 0 0

74INT 0 5

75LOD 1 3

76STO 0 3

77LOD 1 4

78STO 0 4

79LOD 0 3

80LOD 0 4

81OPR 0 9

82JPC 0 100

83LOD 0 3

84LOD 0 4

85OPR 0 10

86JPC 0 91

87LOD 0 4

88LOD 0 3

89OPR 0 3

90STO 0 4

91LOD 0 4

92LOD 0 3

93OPR 0 10

94JPC 0 99

95LOD 0 3

96LOD 0 4

97OPR 0 3

98STO 0 3

99JMP 0 79

100LOD 0 3

101STO 1 5

102OPR 0 0

103INT 0 8

104LIT 0 7

105STO 0 3

106LIT 0 85

107STO 0 4

108CAL 0 2

109LIT 0 25

110STO 0 3

111LIT 0 3

112STO 0 4

113CAL 0 31

114LIT 0 84

115STO 0 3

116LIT 0 36

117STO 0 4

118CAL 0 74

119OPR 0 0

输出的栈中数据(result.txt)

7

85

7

85

0

7

14

42

28

21

35

56

10

112

5

147

224

2

448

1

595

896

0

25

3

25

0

3

6

12

24

48

0

24

1

1

2

12

4

6

8

3

84

36

84

36

48

12

24

12

12

### 第二部分的代码与数据

PL0编译程序源代码（pl0.pas）

program PL0;

{带有代码生成的PL0编译程序}

label 99;

{fpc 编译器不允许跨procedure 的 goto，因此此处 label 没有作用}

const

norw = 13; {保留字的个数}

txmax = 100; {标识符表长度}

nmax = 14; {数字的最大位数}

al = 10; {标识符的长度}

amax = 2047; {最大地址}

levmax = 3; {程序体嵌套的最大深度}

cxmax = 200; {代码数组的大小}

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 );

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

obj = (constant, variable, prcd);

symset = set of symbol;

fct = (lit, opr, lod, sto, cal, int, jmp, jpc, red, wrt); {functions}

instruction = packed record

f : fct; {功能码}

l : 0..levmax; {相对层数}

a : 0..amax; {相对地址}

end;

{LIT 0,a : 取常数a

OPR 0,a : 执行运算a

LOD l,a : 取层差为l的层﹑相对地址为a的变量

STO l,a : 存到层差为l的层﹑相对地址为a的变量

CAL l,a : 调用层差为l的过程

INT 0,a : t寄存器增加a

JMP 0,a : 转移到指令地址a处

JPC 0,a : 条件转移到指令地址a处 }

var

ch : char; {最近读到的字符}

sym : symbol; {最近读到的符号}

id : alfa; {最近读到的标识符}

num : integer; {最近读到的数}

cc : integer; {当前行的字符计数}

ll : integer; {当前行的长度}

kk, err : integer;

cx : integer; {代码数组的当前下标}

line : array [1..81] of char;

a : alfa;

code : array [0..cxmax] of instruction;

word : array [1..norw] of alfa;

wsym : array [1..norw] of symbol;

ssym : array [char] of symbol;

mnemonic : array [fct] of packed array [1..5] of char;

declbegsys, statbegsys, facbegsys : symset;

srcfilename, itmdfilename, resfilename: string;

fin, fitmd, fres: text;

table : array [0..txmax] of

record

name : alfa;

case kind : obj of

constant : (val : integer);

variable, prcd : (level, adr : integer)

end;

procedure error (n : integer);

begin

writeln('\*\*\*\*', ' ' : cc-1, '↑', n : 2); err := err + 1

end {error};

procedure getsym;

var i, j, k : integer;

procedure getch ;

begin

if cc = ll then

begin

if eof(fin) then

begin

write('PROGRAM INCOMPLETE');

close(fin);

close(fitmd);

close(fres);

exit;

end;

ll := 0; cc := 0;

while not eoln(fin) do

begin

ll := ll + 1;

read(fin, ch);

line[ll] := ch;

end;

readln(fin);

ll := ll + 1;

line[ll] := ' ';

end;

cc := cc + 1; ch := line[cc]

end {getch};

begin {getsym}

while ch = ' ' do getch;

if ch in ['a'..'z'] then

begin {标识符或保留字}

k := 0;

repeat

if k < al then

begin

k:= k + 1;

a[k] := ch

end;

getch

until not (ch in ['a'..'z', '0'..'9']);

if k >= kk then kk := k else

repeat a[kk] := ' ';

kk := kk-1

until kk = k;

id := a; i := 1; j := norw;

repeat k := (i+j) div 2;

if id <= word[k] then j := k-1;

if id >= word[k] then i := k + 1

until i > j;

if i-1 > j then begin

sym := wsym[k];

end

else begin

sym := ident;

end;

end

else

if ch in ['0'..'9'] then

begin {数字}

k := 0;

num := 0;

sym := number;

repeat

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

k := k + 1;

getch;

until not (ch in ['0'..'9']);

if k > nmax then

error(30)

end

else

if ch = ':' then

begin

getch;

if ch = '=' then

begin

sym := becomes;

getch;

end

else

sym := nul;

end else

begin

sym := ssym[ch];

getch;

end

end {getsym};

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

begin

if cx > cxmax then

begin

write('PROGRAM TOO LONG');

close(fin);

close(fitmd);

close(fres);

exit;

end;

with code[cx] do

begin f := x; l := y; a := z

end;

cx := cx + 1

end {gen};

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

begin

if not (sym in s1) then

begin

error(n);

s1 := s1 + s2;

while not (sym in s1) do

getsym;

end

end {test};

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

var

dx : integer; {本过程数据空间分配下标}

tx0 : integer; {本过程标识表起始下标}

cx0 : integer; {本过程代码起始下标}

procedure enter(k : obj);

begin {把obj填入符号表中}

tx := tx +1;

with table[tx] do

begin

name := id;

kind := k;

case k of

constant : begin

if num > amax then

begin

error(30);

num := 0

end;

val := num

end;

variable : begin

level := lev;

adr := dx;

dx := dx +1;

end;

prcd : level := lev

end

end

end {enter};

function position(id : alfa) : integer;

var i : integer;

begin {在标识符表中查标识符id}

table[0].name := id; i := tx;

while table[i].name <> id do

i := i-1;

position := i

end {position};

procedure constdeclaration;

begin

if sym = ident then

begin getsym;

if sym in [eql, becomes] then

begin

if sym = becomes then error(1);

getsym;

if sym = number then

begin

enter(constant);

getsym;

end

else error(2);

end else error(3);

end else error(4);

end {constdeclaration};

procedure vardeclaration;

begin

if sym = ident then

begin enter(variable); getsym

end else error(4)

end {vardeclaration};

procedure listcode;

var i : integer;

begin {列出本程序体生成的代码}

For i := cx0 To cx-1 Do

With code[i] Do

writeln(fitmd, i, mnemonic[f] : 5, l : 3, a : 5)

end {listcode};

procedure statement(fsys : symset);

var i, cx1, cx2 : integer;

procedure expression(fsys : symset);

var addop : symbol;

procedure term(fsys : symset);

var mulop : symbol;

procedure factor(fsys : symset);

var i : integer;

begin

test(facbegsys, fsys, 24);

while sym in facbegsys do

begin

if sym = ident then

begin

i := position(id);

if i = 0 then error(11) else

with table[i] do

case kind of

constant : gen(lit, 0, val);

variable : gen(lod, lev-level, adr);

prcd : error(21)

end;

getsym;

end else

if sym = number then

begin

if num > amax then

begin

error(30);

num := 0

end;

gen(lit, 0, num);

getsym;

end

else

if sym = lparen then

begin

getsym;

expression([rparen]+fsys);

if sym = rparen then getsym

else error(22)

end;

test(fsys, [lparen], 23)

end

end {factor};

begin {term}

factor(fsys+[times, slash]);

while sym in [times, slash] do

begin

mulop := sym;

getsym;

factor(fsys+[times, slash]);

if mulop = times then gen(opr, 0, 4)

else gen(opr, 0, 5)

end

end {term};

begin {expression}

if sym in [plus, minus] then

begin

addop := sym;

getsym;

term(fsys+[plus, minus]);

if addop = minus then gen(opr, 0, 1)

end

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

while sym in [plus, minus] do

begin

addop := sym;

getsym;

term(fsys+[plus, minus]);

if addop = plus then gen(opr, 0, 2)

else gen(opr, 0, 3)

end

end {expression};

procedure condition(fsys : symset);

var relop : symbol;

begin

if sym = oddsym then

begin

getsym;

expression(fsys);

gen(opr, 0, 6)

end else

begin

expression([eql, neq, lss, gtr, leq, geq] + fsys);

if not (sym in [eql, neq, lss, leq, gtr, geq]) then

error(20) else

begin

relop := sym;

getsym;

expression(fsys);

case relop of

eql : gen(opr, 0, 8);

neq : gen(opr, 0, 9);

lss : gen(opr, 0, 10);

geq : gen(opr, 0, 11);

gtr : gen(opr, 0, 12);

leq : gen(opr, 0, 13);

end

end

end

end {condition};

begin {statement}

if sym = ident then

begin i := position(id);

if i = 0 then error(11) else

if table[i].kind <> variable then

begin {对非变量赋值}

error(12);

i := 0;

end;

getsym;

if sym = becomes then

getsym

else error(13);

expression(fsys);

if i <> 0 then

with table[i] do gen(sto, lev-level, adr);

end else

if sym = callsym then

begin getsym;

if sym <> ident then error(14) else

begin

i := position(id);

if i = 0 then error(11) else

with table[i] do

if kind = prcd then

gen(cal, lev-level, adr)

else error(15);

getsym;

end

end else

if sym = ifsym then

begin

getsym;

condition([thensym, dosym]+fsys);

if sym = thensym then

getsym

else error(16);

cx1 := cx;

gen(jpc, 0, 0);

statement(fsys);

code[cx1].a := cx

end else

if sym = beginsym then

begin

getsym;

statement([semicolon, endsym]+fsys);

while sym in [semicolon]+statbegsys do

begin

if sym = semicolon then

getsym

else error(10);

statement([semicolon, endsym]+fsys)

end;

if sym = endsym then

getsym

else error(17)

end else

if sym = whilesym then

begin

cx1 := cx;

getsym;

condition([dosym]+fsys);

cx2 := cx;

gen(jpc, 0, 0);

if sym = dosym then

getsym

else error(18);

statement(fsys);

gen(jmp, 0, cx1);

code[cx2].a := cx

end else

if sym = readsym then

begin

getsym;

if sym = lparen then

repeat

getsym;

if sym = ident then

begin

i := position(id);

if i = 0 then

error(33)

else if table[i].kind <> variable then

error(34)

else with table[i] do

gen(red, lev - level, adr);

end

else error(35);

{读取read 中的每个变量，以逗号为分隔}

getsym;

until sym <> comma

else error(36);

{右括号结束}

if sym <> rparen then error(37);

getsym;

end else

if sym = writesym then

begin

getsym;

if sym = lparen then

begin

repeat

getsym;

expression([rparen, comma] + fsys);

with table[i] do

gen(wrt, lev - level, adr);

until sym <> comma;

if sym <> rparen then

error(39);

getsym;

end

else error(38);

end;

test(fsys, [ ], 19);

end {statement};

begin {block}

dx := 3;

tx0 := tx;

table[tx].adr := cx;

gen(jmp, 0, 0);

if lev > levmax then error(32);

repeat

if sym = constsym then

begin getsym;

repeat

constdeclaration;

while sym = comma do

begin

getsym;

constdeclaration;

end;

if sym = semicolon then

getsym

else error(5)

until sym <> ident

end;

if sym = varsym then

begin

getsym;

repeat

vardeclaration;

while sym = comma do

begin

getsym;

vardeclaration;

end;

if sym = semicolon then

getsym

else error(5);

until sym <> ident;

end;

while sym = procsym do

begin getsym;

if sym = ident then

begin

enter(prcd);

getsym;

end

else error(4);

if sym = semicolon then

getsym

else error(5);

block(lev+1, tx, [semicolon]+fsys);

if sym = semicolon then

begin

getsym;

test(statbegsys+[ident, procsym], fsys, 6)

end

else error(5)

end;

test(statbegsys+[ident], declbegsys, 7)

until not (sym in declbegsys);

code[table[tx0].adr].a := cx;

with table[tx0] do

begin

adr := cx; {代码开始地址}

end;

cx0 := cx;

gen(int, 0, dx);

statement([semicolon, endsym]+fsys);

gen(opr, 0, 0); {生成返回指令}

test(fsys, [ ], 8);

listcode;

end {block};

procedure interpret;

const stacksize = 500;

var p, b, t : integer; {程序地址寄存器, 基地址寄存器,栈顶地址寄存器}

i : instruction; {指令寄存器}

s : array [1..stacksize] of integer; {数据存储栈}

function base(l : integer) : integer;

var b1 : integer;

begin

b1 := b; {顺静态链求层差为l的层的基地址}

while l > 0 do

begin

b1 := s[b1];

l := l-1

end;

base := b1

end {base};

begin

writeln('START PL/0');

t := 0; b := 1; p := 0;

s[1] := 0; s[2] := 0; s[3] := 0;

repeat

i := code[p];

p := p+1;

with i do

case f of

lit : begin

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

end;

opr : case a of {运算}

0 : begin {返回}

t := b-1; p := s[t+3]; b := s[t+2];

end;

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

2 : begin

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

end;

3 : begin

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

end;

4 : begin

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

end;

5 : begin

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

end;

6 : s[t] := ord(odd(s[t]));

8 : begin t := t-1;

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

end;

9: begin t := t-1;

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

end;

10 : begin t := t-1;

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

end;

11: begin t := t-1;

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

end;

12 : begin t := t-1;

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

end;

13 : begin t := t-1;

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

end;

end;

lod : begin

t := t + 1; s[t] := s[base(l) + a]

end;

sto : begin

s[base(l) + a] := s[t];

writeln(fres, s[t]);

t := t-1

end;

cal : begin {generate new block mark}

s[t+1] := base( l ); s[t+2] := b;

s[t+3] := p;

b := t+1; p := a

end;

int : t := t + a;

jmp : p := a;

jpc : begin

if s[t] = 0 then p := a;

t := t-1

end;

red : read(s[base(l) + a]);

wrt : writeln(s[t])

end {with, case}

until p = 0;

write('END PL/0');

end {interpret};

begin {主程序}

for ch := 'A' to ';' do ssym[ch] := 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 ';

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;

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

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

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

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

ssym['.'] := period; ssym['~'] := neq;

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

ssym['$'] := leq; ssym['^'] := geq;

ssym[';'] := semicolon;

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 ';

declbegsys := [constsym, varsym, procsym];

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

facbegsys := [ident, number, lparen];

write('Source File: ');

readln(srcfilename);

itmdfilename := 'intermediate.txt';

resfilename := 'result.txt';

assign(fin, srcfilename);

assign(fitmd, itmdfilename);

assign(fres, resfilename);

rewrite(fitmd);

rewrite(fres);

reset(fin);

err := 0;

cc := 0; cx := 0; ll := 0; ch := ' '; kk := al;

getsym;

block(0, 0, [period]+declbegsys+statbegsys);

if sym <> period then error(9);

if err = 0 then interpret

else write('ERRORS IN PL/0 PROGRAM');

close(fin);

close(fitmd);

close(fres);

99 : writeln

end.

PL0源程序代码（source.pl0）

const m = 7, n = 85;

var x, y, z, q, r;

procedure multiply;

var a, b;

begin a := x; b := y; z := 0;

while b > 0 do

begin

if odd b then z := z + a;

a := 2\*a ; b := b/2 ;

end

end;

procedure divide;

var w;

begin r := x; q := 0; w := y;

while w $ r do w := 2\*w ;

while w > y do

begin q := 2\*q; w := w/2;

if w $ r then

begin r := r-w; q := q+1 end

end

end;

procedure gcd;

var f, g ;

begin f := x; g := y;

while f ~ g do

begin

if f < g then g := g-f;

if g < f then f := f-g;

end;

z := f

end;

begin

read(x); read(y); call multiply;

write(x, y, z);

read(x); read(y); call divide;

write(x, y, q);

read(x); read(y); call gcd;

write(x, y, z);

end.

中间代码（intermediate.txt）

2INT 0 5

3LOD 1 3

4STO 0 3

5LOD 1 4

6STO 0 4

7LIT 0 0

8STO 1 5

9LOD 0 4

10LIT 0 0

11OPR 0 12

12JPC 0 29

13LOD 0 4

14OPR 0 6

15JPC 0 20

16LOD 1 5

17LOD 0 3

18OPR 0 2

19STO 1 5

20LIT 0 2

21LOD 0 3

22OPR 0 4

23STO 0 3

24LOD 0 4

25LIT 0 2

26OPR 0 5

27STO 0 4

28JMP 0 9

29OPR 0 0

31INT 0 4

32LOD 1 3

33STO 1 7

34LIT 0 0

35STO 1 6

36LOD 1 4

37STO 0 3

38LOD 0 3

39LOD 1 7

40OPR 0 13

41JPC 0 47

42LIT 0 2

43LOD 0 3

44OPR 0 4

45STO 0 3

46JMP 0 38

47LOD 0 3

48LOD 1 4

49OPR 0 12

50JPC 0 72

51LIT 0 2

52LOD 1 6

53OPR 0 4

54STO 1 6

55LOD 0 3

56LIT 0 2

57OPR 0 5

58STO 0 3

59LOD 0 3

60LOD 1 7

61OPR 0 13

62JPC 0 71

63LOD 1 7

64LOD 0 3

65OPR 0 3

66STO 1 7

67LOD 1 6

68LIT 0 1

69OPR 0 2

70STO 1 6

71JMP 0 47

72OPR 0 0

74INT 0 5

75LOD 1 3

76STO 0 3

77LOD 1 4

78STO 0 4

79LOD 0 3

80LOD 0 4

81OPR 0 9

82JPC 0 100

83LOD 0 3

84LOD 0 4

85OPR 0 10

86JPC 0 91

87LOD 0 4

88LOD 0 3

89OPR 0 3

90STO 0 4

91LOD 0 4

92LOD 0 3

93OPR 0 10

94JPC 0 99

95LOD 0 3

96LOD 0 4

97OPR 0 3

98STO 0 3

99JMP 0 79

100LOD 0 3

101STO 1 5

102OPR 0 0

103INT 0 8

104RED 0 3

105RED 0 4

106CAL 0 2

107LOD 0 3

108WRT 0 2

109LOD 0 4

110WRT 0 2

111LOD 0 5

112WRT 0 2

113RED 0 3

114RED 0 4

115CAL 0 31

116LOD 0 3

117WRT 0 31

118LOD 0 4

119WRT 0 31

120LOD 0 6

121WRT 0 31

122RED 0 3

123RED 0 4

124CAL 0 74

125LOD 0 3

126WRT 0 74

127LOD 0 4

128WRT 0 74

129LOD 0 5

130WRT 0 74

131OPR 0 0

输入输出数据

（输入）

123 20

（输出）

123

20

2460

（输入）

500 20

（输出）

500

20

25

（输入）

15 10

（输出）

15

10

5