Pl0 源程序实验

学号：16340150

姓名：刘俊峰

# Part1

编译源程序：

*// program PL0 ( fin, output);*

program pl0;

const

norw = 11;

txmax = 100;

nmax = 14;

al = 10;

amax = 2047;

levmax = 3;

cxmax = 200;

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;

objecttyp = (constant, variable, prosedure);

symset = set of symbol;

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

instruction = packed record

f : fct;

l : 0..levmax;

a : 0..amax;

end;

var

fin, fout: text;

sfile, dfile: string;

ch : char;

sym : symbol;

id : alfa;

num : integer;

cc : integer;

ll : integer;

kk, err : integer;

cx : integer;

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;

table : array [0..txmax] of

record

name : alfa;

case kind : objecttyp of

constant : (val : integer);

variable, prosedure : (level, adr : integer)

end;

procedure error (n : integer);

begin

writeln( '\*\*\*\*', ' ':cc-1, '^', n:2 );

writeln(fout, '\*\*\*\*', ' ' : cc-1, '^', n : 2);

err := err + 1

end;

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);

exit;

end;

ll := 0;

cc := 0;

write(cx : 5, ' ');

write(fout, cx: 5, ' ');

while not eoln(fin) do

begin

ll := ll + 1;

read(fin, ch);

write(ch);

write(fout, ch);

line[ll] := ch

end;

writeln;

writeln(fout);

readln(fin);

ll := ll + 1;

line[ll] := ' '

end;

cc := cc + 1;

ch := line[cc]

end;

begin

while ch = ' ' do getch;

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

begin

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

sym := wsym[k]

else sym := ident

end

else

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

begin

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

if ch = '<' then

begin

getch;

if ch = '=' then

begin

sym := leq;

getch

end

else

if ch = '>' then

begin

sym := neq;

getch

end

else sym := lss

end

else if ch = '>' then

begin

getch;

if ch = '=' then

begin

sym := geq;

getch

end

else sym := gtr

end

else

begin

sym := ssym[ch];

getch

end

end;

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

begin

if cx > cxmax then

begin

write('PROGRAM TOO LONG');

close(fin);

exit

end;

with code[cx] do

begin

f := x;

l := y;

a := z

end;

cx := cx + 1

end

;

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;

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

var

dx : integer;

tx0 : integer;

cx0 : integer;

procedure enter(k : objecttyp);

begin

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;

prosedure : level := lev

end

end

end;

function position(id : alfa) : integer;

var i : integer;

begin

table[0].name := id;

i := tx;

while table[i].name <> id do

i := i-1;

position := i

end;

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;

procedure vardeclaration;

begin

if sym = ident then

begin

enter(variable);

getsym

end

else error(4)

end

;

procedure listcode;

var i : integer;

begin

for i := cx0 to cx-1 do

with code[i] do

begin

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

writeln(fout,i:4,mnemonic[f]:7,l:3,a:5);

end;

end;

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);

prosedure : 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;

begin

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

;

begin

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;

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;

begin

if sym = ident then

begin

i := position(id);

if i = 0 then

error(11)

else

if table[i].kind <> variable then

begin

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 = prosedure 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;

begin

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(prosedure);

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;

end;

cx0 := cx;

gen(int, 0, dx);

statement([semicolon, endsym]+fsys);

gen(opr, 0, 0);

test(fsys, [ ], 8);

listcode;

end;

procedure interpret;

const stacksize = 500;

var p, b, t : integer;

i : instruction;

s : array [1..stacksize] of integer;

function base(l : integer) : integer;

var b1 : integer;

begin

b1 := b;

while l > 0 do

begin

b1 := s[b1];

l := l-1

end;

base := b1

end;

begin

writeln('START PL/0');

writeln(fout,'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

0 : *//*

begin

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(s[t]);

writeln(fout, s[t]);

t := t-1

end;

cal : *//*

begin

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

until p = 0;

writeln('END PL/0');

writeln(fout,'END PL/0');

end;

begin

writeln('please input source program file name : ');

readln(sfile);

assign(fin,sfile);

reset(fin);

writeln('please input the file name to save result : ');

readln(dfile);

assign(fout,dfile);

rewrite(fout);

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

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];

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');

writeln;

close(fin);

readln(sfile);

close(fout);

end.

PL0源程序：

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.

结果：

0 const m = 7, n = 85;

1 var x, y, z, q, r;

1

1 procedure multiply;

1 var a, b;

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

9 while b > 0 do

13 begin

13 if odd b then z := z + a;

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

28 end;

29 end;

2 INT 0 5

3 LOD 1 3

4 STO 0 3

5 LOD 1 4

6 STO 0 4

7 LIT 0 0

8 STO 1 5

9 LOD 0 4

10 LIT 0 0

11 OPR 0 12

12 JPC 0 29

13 LOD 0 4

14 OPR 0 6

15 JPC 0 20

16 LOD 1 5

17 LOD 0 3

18 OPR 0 2

19 STO 1 5

20 LIT 0 2

21 LOD 0 3

22 OPR 0 4

23 STO 0 3

24 LOD 0 4

25 LIT 0 2

26 OPR 0 5

27 STO 0 4

28 JMP 0 9

29 OPR 0 0

30

30 procedure divide;

30 var w;

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

38 while w <= r do w := 2\*w ;

47 while w > y do

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

59 if w <= r then

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

71 end;

72 end;

31 INT 0 4

32 LOD 1 3

33 STO 1 7

34 LIT 0 0

35 STO 1 6

36 LOD 1 4

37 STO 0 3

38 LOD 0 3

39 LOD 1 7

40 OPR 0 13

41 JPC 0 47

42 LIT 0 2

43 LOD 0 3

44 OPR 0 4

45 STO 0 3

46 JMP 0 38

47 LOD 0 3

48 LOD 1 4

49 OPR 0 12

50 JPC 0 72

51 LIT 0 2

52 LOD 1 6

53 OPR 0 4

54 STO 1 6

55 LOD 0 3

56 LIT 0 2

57 OPR 0 5

58 STO 0 3

59 LOD 0 3

60 LOD 1 7

61 OPR 0 13

62 JPC 0 71

63 LOD 1 7

64 LOD 0 3

65 OPR 0 3

66 STO 1 7

67 LOD 1 6

68 LIT 0 1

69 OPR 0 2

70 STO 1 6

71 JMP 0 47

72 OPR 0 0

73

73 procedure gcd;

73 var f, g ;

74 begin f := x; g := y;

79 while f <> g do

83 begin

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

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

99 end;

100 z := f;

102 end;

74 INT 0 5

75 LOD 1 3

76 STO 0 3

77 LOD 1 4

78 STO 0 4

79 LOD 0 3

80 LOD 0 4

81 OPR 0 9

82 JPC 0 100

83 LOD 0 3

84 LOD 0 4

85 OPR 0 10

86 JPC 0 91

87 LOD 0 4

88 LOD 0 3

89 OPR 0 3

90 STO 0 4

91 LOD 0 4

92 LOD 0 3

93 OPR 0 10

94 JPC 0 99

95 LOD 0 3

96 LOD 0 4

97 OPR 0 3

98 STO 0 3

99 JMP 0 79

100 LOD 0 3

101 STO 1 5

102 OPR 0 0

103

103 begin

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

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

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

119 end.

103 INT 0 8

104 LIT 0 7

105 STO 0 3

106 LIT 0 85

107 STO 0 4

108 CAL 0 2

109 LIT 0 25

110 STO 0 3

111 LIT 0 3

112 STO 0 4

113 CAL 0 31

114 LIT 0 84

115 STO 0 3

116 LIT 0 36

117 STO 0 4

118 CAL 0 74

119 OPR 0 0

START PL/0

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

END PL/0

# Part2

编译程序：

*// program PL0 ( fin, output);*

program pl0;

const

norw = 13;

txmax = 100;

nmax = 14;

al = 10;

amax = 2047;

levmax = 3;

cxmax = 200;

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;

objecttyp = (constant, variable, prosedure);

symset = set of symbol;

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

instruction = packed record

f : fct;

l : 0..levmax;

a : 0..amax;

end;

var

fin, fout: text;

sfile, dfile: string;

ch : char;

sym : symbol;

id : alfa;

num : integer;

cc : integer;

ll : integer;

kk, err : integer;

cx : integer;

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;

table : array [0..txmax] of

record

name : alfa;

case kind : objecttyp of

constant : (val : integer);

variable, prosedure : (level, adr : integer)

end;

procedure error (n : integer);

begin

writeln( '\*\*\*\*', ' ':cc-1, '^', n:2 );

writeln(fout, '\*\*\*\*', ' ' : cc-1, '^', n : 2);

err := err + 1

end;

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);

exit;

end;

ll := 0;

cc := 0;

write(cx : 5, ' ');

write(fout, cx: 5, ' ');

while not eoln(fin) do

begin

ll := ll + 1;

read(fin, ch);

write(ch);

write(fout, ch);

line[ll] := ch

end;

writeln;

writeln(fout);

readln(fin);

ll := ll + 1;

line[ll] := ' '

end;

cc := cc + 1;

ch := line[cc]

end;

begin

while ch = ' ' do getch;

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

begin

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

sym := wsym[k]

else sym := ident

end

else

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

begin

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

if ch = '<' then

begin

getch;

if ch = '=' then

begin

sym := leq;

getch

end

else

if ch = '>' then

begin

sym := neq;

getch

end

else sym := lss

end

else if ch = '>' then

begin

getch;

if ch = '=' then

begin

sym := geq;

getch

end

else sym := gtr

end

else

begin

sym := ssym[ch];

getch

end

end;

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

begin

if cx > cxmax then

begin

write('PROGRAM TOO LONG');

close(fin);

exit

end;

with code[cx] do

begin

f := x;

l := y;

a := z

end;

cx := cx + 1

end

;

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;

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

var

dx : integer;

tx0 : integer;

cx0 : integer;

procedure enter(k : objecttyp);

begin

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;

prosedure : level := lev

end

end

end;

function position(id : alfa) : integer;

var i : integer;

begin

table[0].name := id;

i := tx;

while table[i].name <> id do

i := i-1;

position := i

end;

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;

procedure vardeclaration;

begin

if sym = ident then

begin

enter(variable);

getsym

end

else error(4)

end

;

procedure listcode;

var i : integer;

begin

for i := cx0 to cx-1 do

with code[i] do

begin

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

writeln(fout,i:4,mnemonic[f]:7,l:3,a:5);

end;

end;

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);

prosedure : 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;

begin

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

;

begin

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;

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;

begin

if sym = ident then

begin

i := position(id);

if i = 0 then

error(11)

else

if table[i].kind <> variable then

begin

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 = prosedure 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(11)

else if table[i].kind <> variable

then

begin

error(12);

i := 0

end

else with table[i] Do

gen(red,lev-level,adr)

end

else error(4);

getsym;

until sym <> comma

else error(40);

if sym <> rparen

then error(22);

getsym

end

else if sym = writesym

then

begin

getsym;

if sym = lparen

then

begin

repeat

getsym;

expression([rparen,comma]+fsys);

gen(wrt,0,0);

until sym <> comma;

if sym <> rparen

then error(22);

getsym

end

else error(40)

end;

test(fsys, [ ], 19)

end;

begin

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(prosedure);

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;

end;

cx0 := cx;

gen(int, 0, dx);

statement([semicolon, endsym]+fsys);

gen(opr, 0, 0);

test(fsys, [ ], 8);

listcode;

end;

procedure interpret;

const stacksize = 500;

var p, b, t : integer;

i : instruction;

s : array [1..stacksize] of integer;

function base(l : integer) : integer;

var b1 : integer;

begin

b1 := b;

while l > 0 do

begin

b1 := s[b1];

l := l-1

end;

base := b1

end;

begin

writeln('START PL/0');

writeln(fout,'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

0 : *//*

begin

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(s[t]);

writeln(fout, s[t]);

t := t-1

end;

cal : *//*

begin

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 :

begin

writeln('Input a integer:');

writeln(fout,'Input a integer:');

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

writeln(fout,s[base(l)+a]);

end;

wrt :

begin

writeln('Here is the integer:');

writeln(s[t]);

writeln(fout,'Here is the integer:');

writeln(fout,s[t]);

t := t+1

end

end

until p = 0;

writeln('END PL/0');

writeln(fout,'END PL/0');

end;

begin

writeln('please input source program file name : ');

readln(sfile);

assign(fin,sfile);

reset(fin);

writeln('please input the file name to save result : ');

readln(dfile);

assign(fout,dfile);

rewrite(fout);

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

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];

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');

writeln;

close(fin);

readln(sfile);

close(fout);

end.

PL0源程序：

procedure test;

var input;

begin

read(input);

write(input);

end;

begin

call test;

end.

结果：

0 procedure test;

1 var input;

2 begin

3 read(input);

4 write(input);

6 end;

2 INT 0 4

3 RED 0 3

4 LOD 0 3

5 WRT 0 0

6 OPR 0 0

7

7 begin

8 call test;

9 end.

7 INT 0 3

8 CAL 0 2

9 OPR 0 0

START PL/0

Input a integer:

98

Here is the integer:

98

END PL/0