PL0 编译程序

```
program  PL0 ( input, output);
{带有代码生成的 PL0 编译程序}
label  99;
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;
   object = (constant, variable, procedure);
   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;
   table : array [0..txmax] of
              record
                 name : alfa;
                 case kind : object of
                    constant : (val : integer);
                    variable, procedure : (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(input) then
      begin
        write(‘PROGRAM INCOMPLETE’); goto 99
      end;
      ll := 0; cc := 0; write(cx : 5, ‘ ‘);
      while  ¬eoln(input) do
      begin
        ll := ll + 1; read(ch); write(ch);
        line[ll] := ch
      end;
      writeln; ll := ll + 1; read(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  ¬(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 {数字}
    k := 0;    num := 0;    sym := number;
    repeat
        num := 10*num + (ord(ch)－ord(0));
        k := k + 1;    getch;
    until  ¬(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’); goto 99
    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 ¬(sym in s1) then
  begin   error(n);   s1 := s1 + s2;
    while ¬(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 : object);
  begin {把 object 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;
        procedure : 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(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);l
              procedure : 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 ¬(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 = procedure 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(procedure);   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 ¬(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(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];
    page(output); 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’);
99 : writeln
end.
```