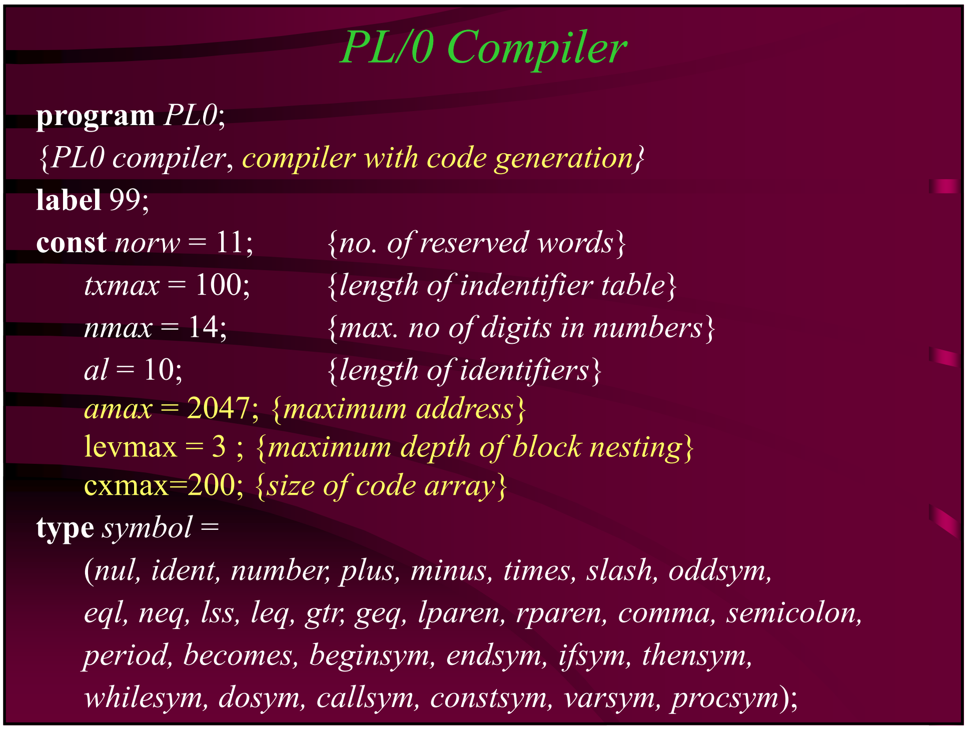
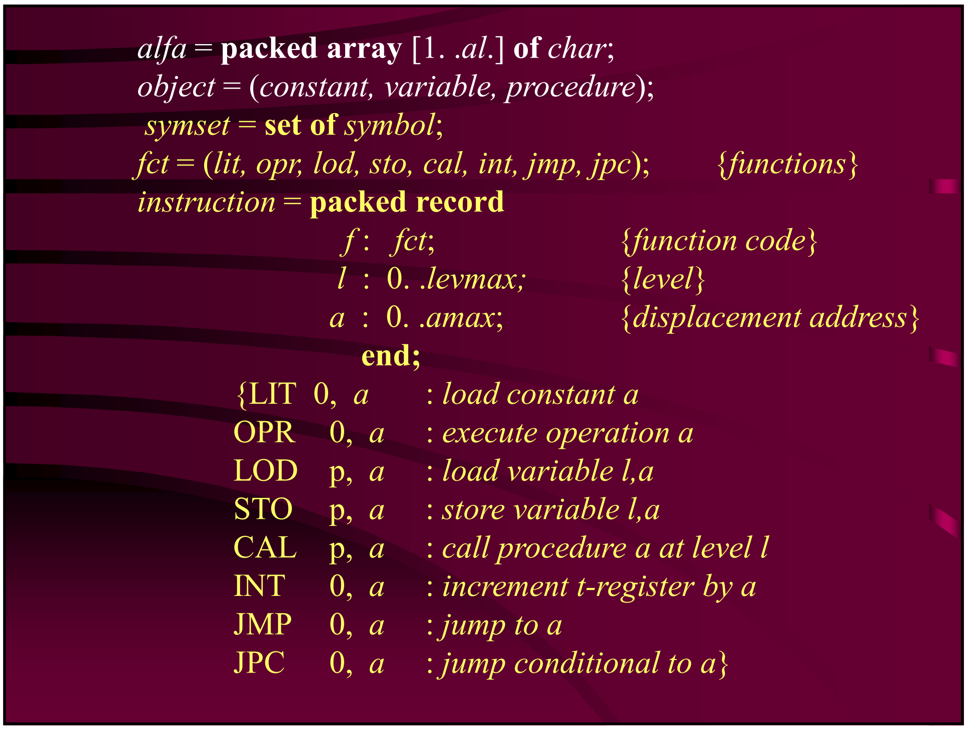
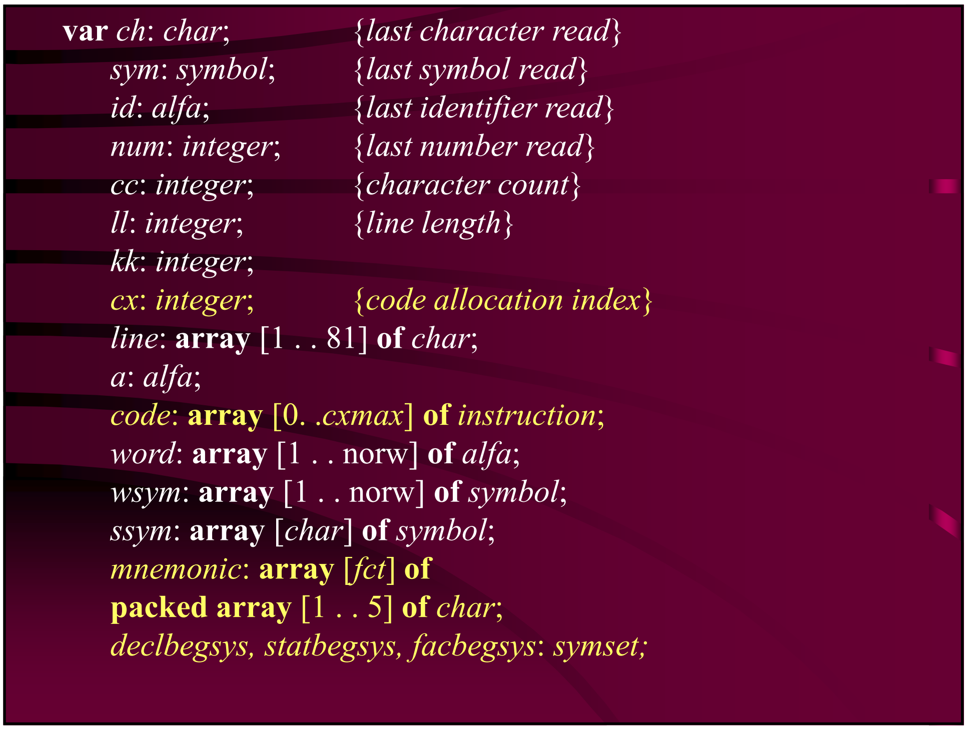


PL/0 Compiler

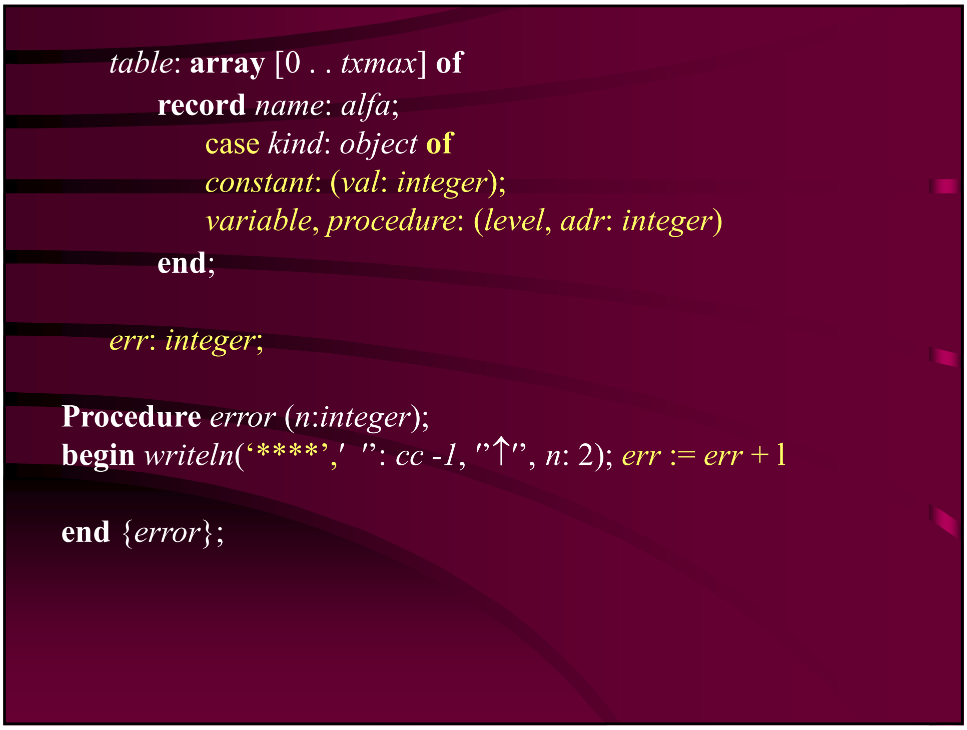
```
program PL0;
{PL0 compiler, compiler with code generation}
label 99;
const norw = 11;        {no. of reserved words}
    txmax = 100;        {length of indentifier table}
    nmax = 14;          {max. no of digits in numbers}
    al = 10;            {length of identifiers}
    amax = 2047; {maximum address}
    levmax = 3 ; {maximum depth of block nesting}
    cxmax=200; {size of code array}
type symbol =
    (nul, ident, number, plus, minus, times, slash, oddsym,
    eql, neq, lss, leq, gtr, geq, lparen, rparen, comma, semicolon,
    period, becomes, beginsym, endsym, ifsym, thensym,
    whilesym, dosym, callsym, constsym, varsym, procsym);
```



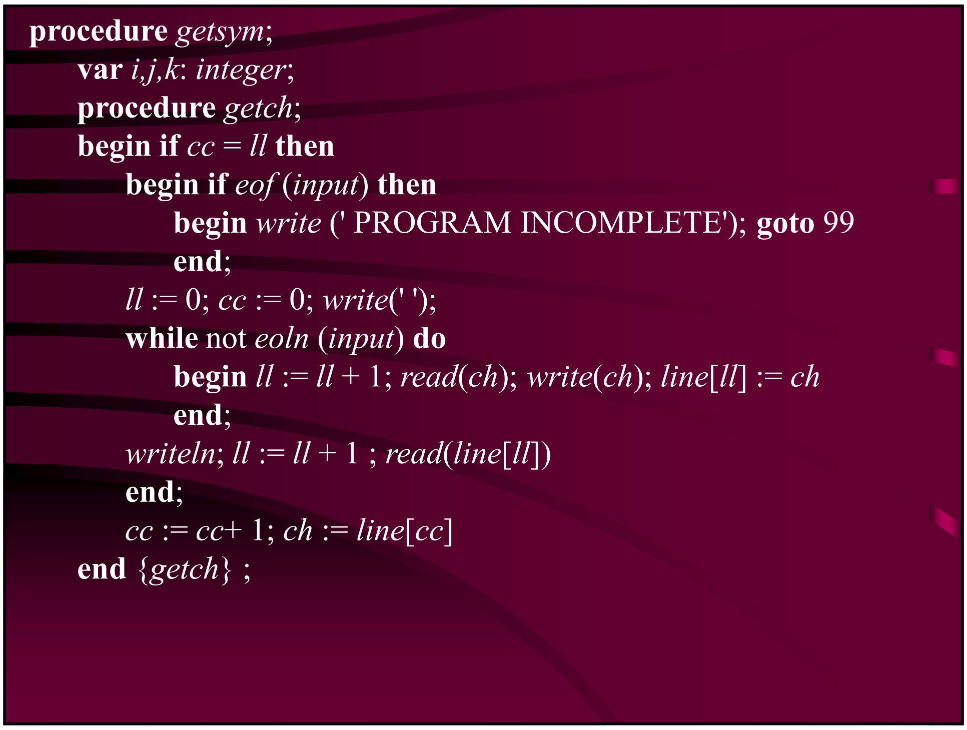
```
        alfa = packed array [1. .al.] of char;
        object = (constant, variable, procedure);
         symset = set of symbol;
        fct = (lit, opr, lod, sto, cal, int, jmp, jpc);       {functions}
        instruction = packed record
                            f :  fct;                  {function code}
                            l  :  0. .levmax;          {level}
                            a  :  0. .amax;            {displacement address}
                               end;
                {LIT  0,  a      : load constant a
                OPR    0,  a     : execute operation a
                LOD    p,  a     : load variable l,a
                STO    p,  a     : store variable l,a
                CAL    p,  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}
```



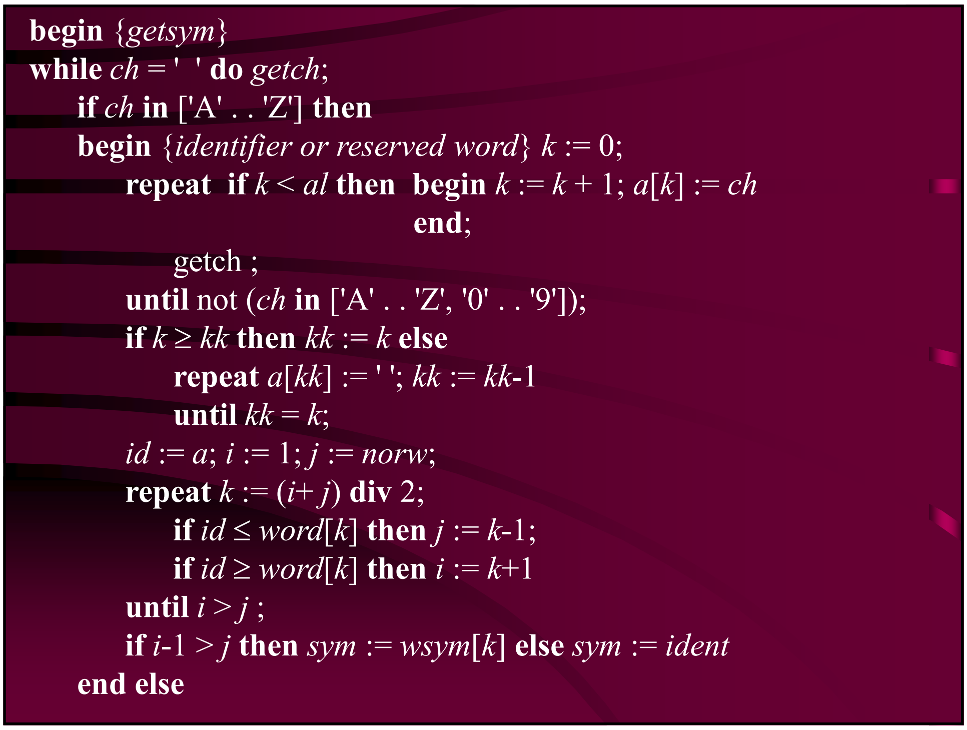
```
var ch: char;                {last character read}
    sym: symbol;             {last symbol read}
    id: alfa;                {last identifier read}
    num: integer;            {last number read}
    cc: integer;             {character count}
    ll: integer;             {line length}
    kk: integer;
    cx: integer;             {code allocation index}
    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;
```



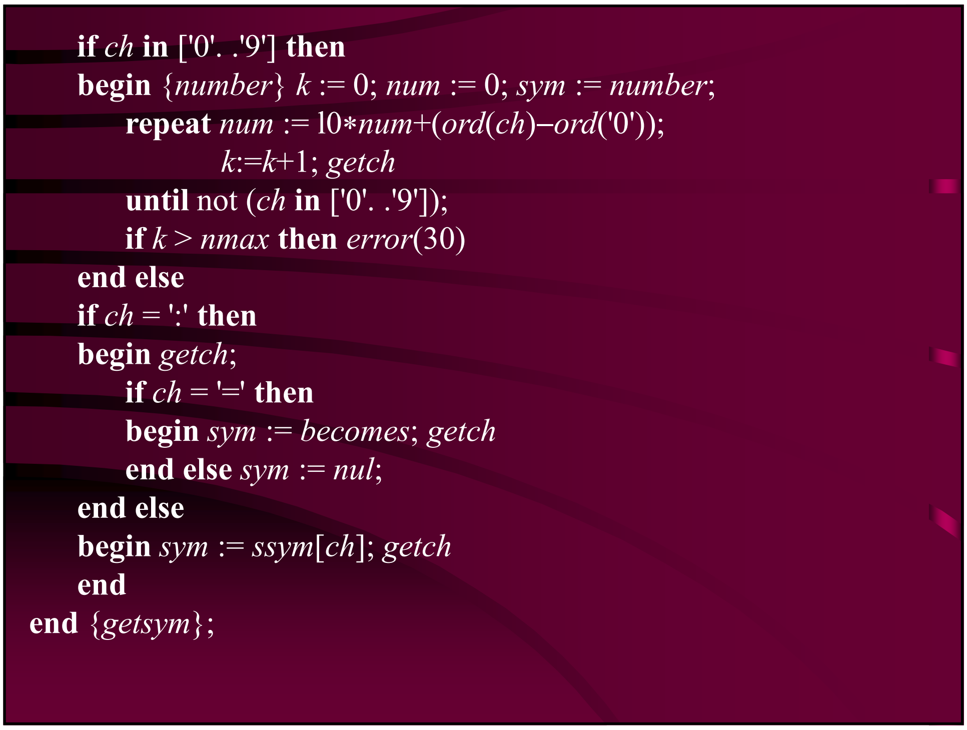
```
    table: array [0 . . txmax] of
        record name: alfa;
            case kind: object of
            constant: (val: integer);
            variable, procedure: (level, adr: integer)
        end;

    err: integer;

Procedure error (n:integer);
begin writeln(‘****’,'  ”: cc -1, ”↑”, n: 2); err := err + 1

end {error};
```



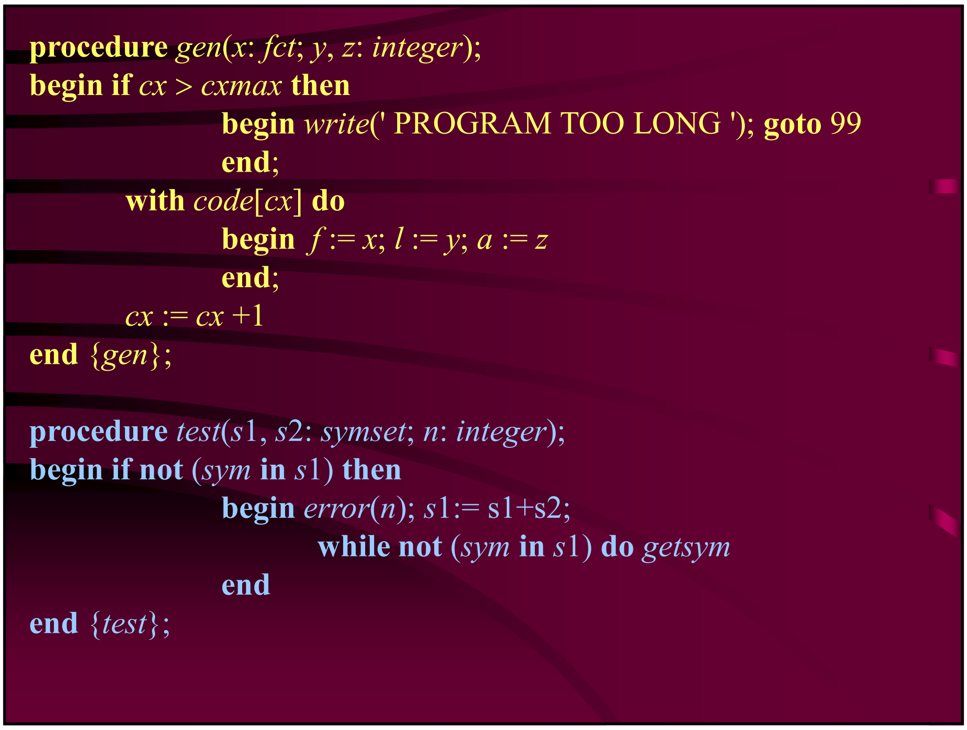
```
procedure getsym;
    var i,j,k: integer;
    procedure getch;
    begin if cc = ll then
        begin if eof (input) then
            begin write (' PROGRAM INCOMPLETE'); goto 99
            end;
        ll := 0; cc := 0; write(' ');
        while not eoln (input) do
            begin ll := ll + 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} ;
```



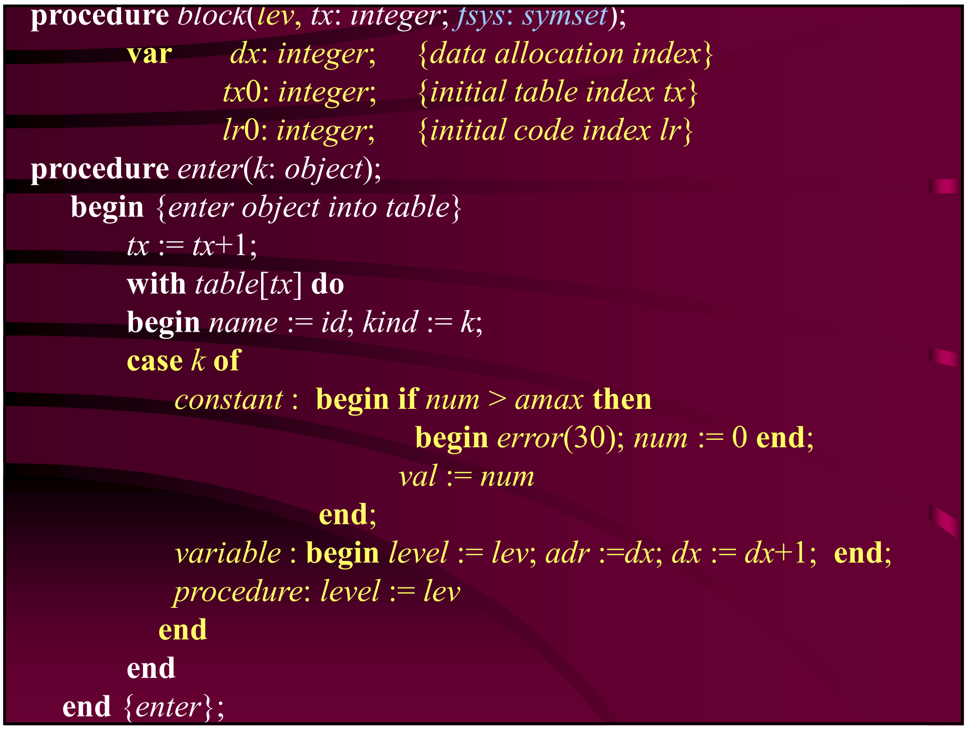
```
begin {getsym}
while ch = ' ' do getch;
    if ch in ['A' . . 'Z'] then
    begin {identifier or reserved word} k := 0;
        repeat  if k < al then  begin k := k + 1; a[k] := ch
                                end;
            getch ;
        until not (ch in ['A' . . 'Z', '0' . . '9']);
        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
```



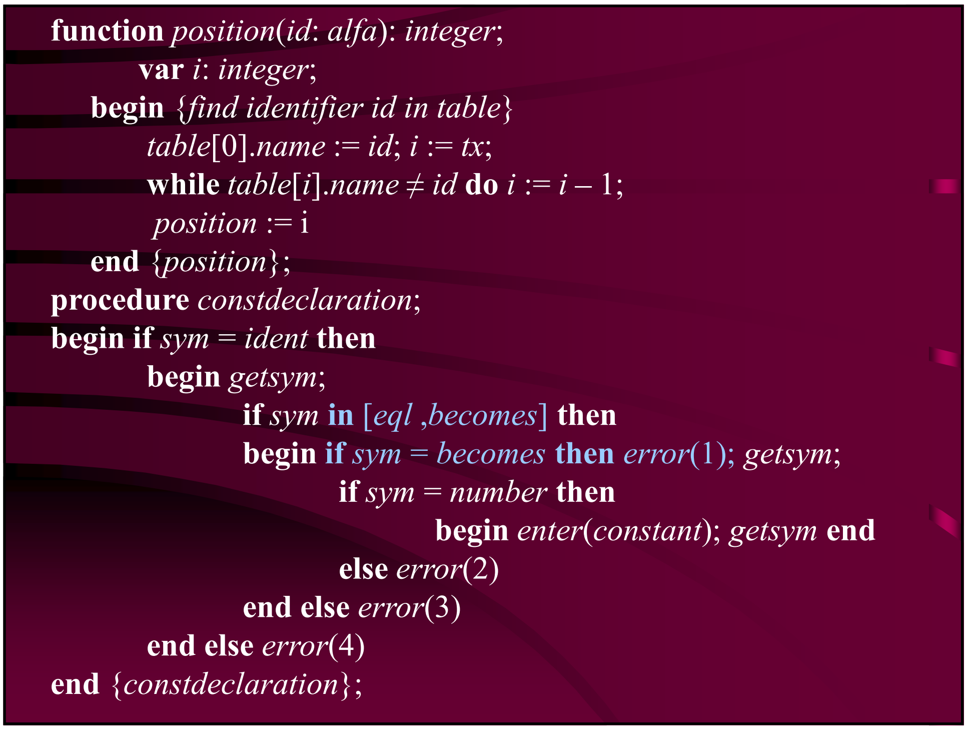
```
    if ch in ['0'. .'9'] then
    begin {number} k := 0; num := 0; sym := number;
        repeat num := 10∗num+(ord(ch)−ord('0'));
                k:=k+1; getch
        until not (ch in ['0'. .'9']);
        if k > nmax then error(30)
    end else
    if ch = ':' then
    begin getch;
        if ch = '=' then
        begin sym := becomes; getch
        end else sym := nul;
    end else
    begin sym := ssym[ch]; getch
    end
end {getsym};
```



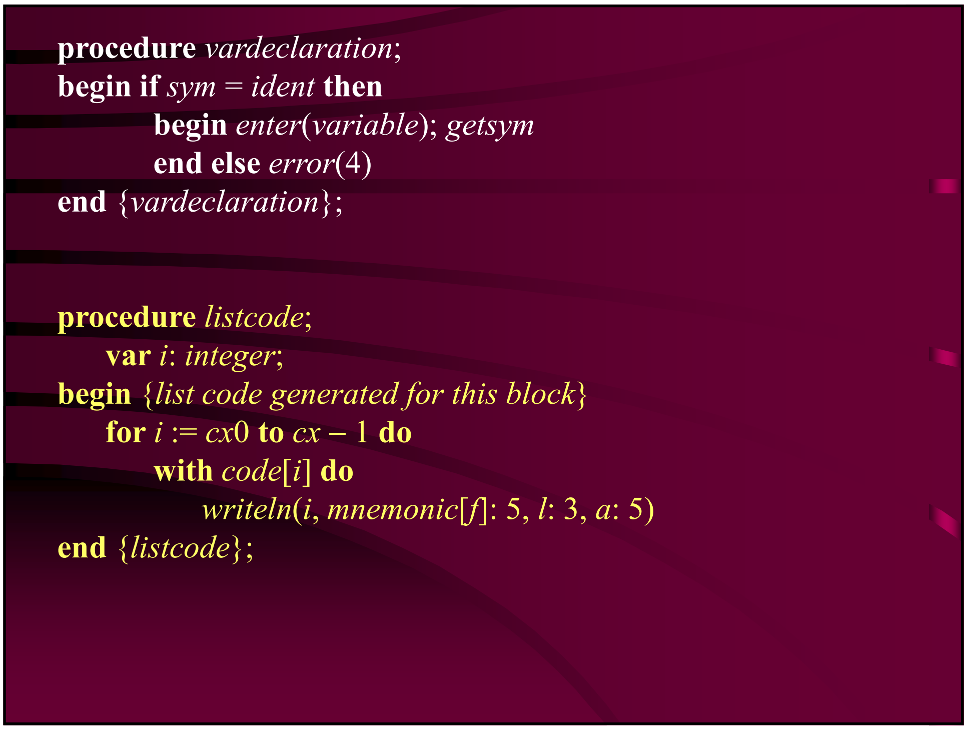
```
procedure gen(x: fct; y, z: integer);
begin if cx > cxmax then
                begin write(' PROGRAM TOO LONG '); goto 99
                end;
        with code[cx] do
                begin  f := x; l := y; a := z
                end;
        cx := cx +1
end {gen};

procedure test(s1, s2: symset; n: integer);
begin if not (sym in s1) then
                begin error(n); s1:= s1+s2;
                        while not (sym in s1) do getsym
                end
end {test};
```



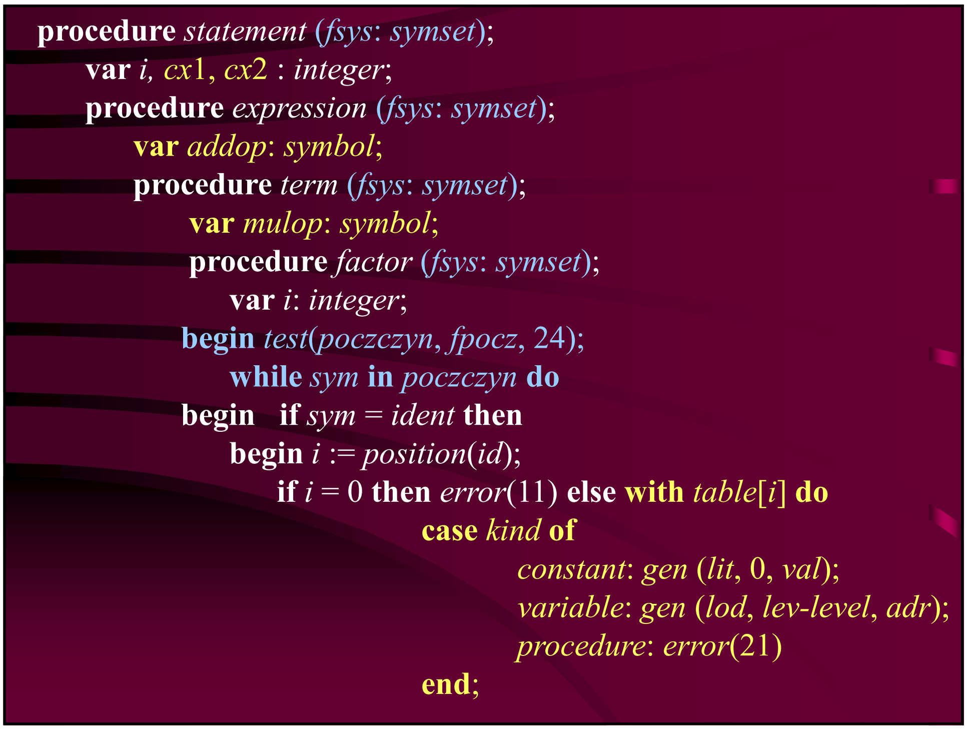
```
procedure block(lev, tx: integer; fsys: symset);
        var       dx: integer;     {data allocation index}
                  tx0: integer;    {initial table index tx}
                  lr0: integer;    {initial code index lr}
procedure enter(k: object);
    begin {enter object into table}
        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};
```



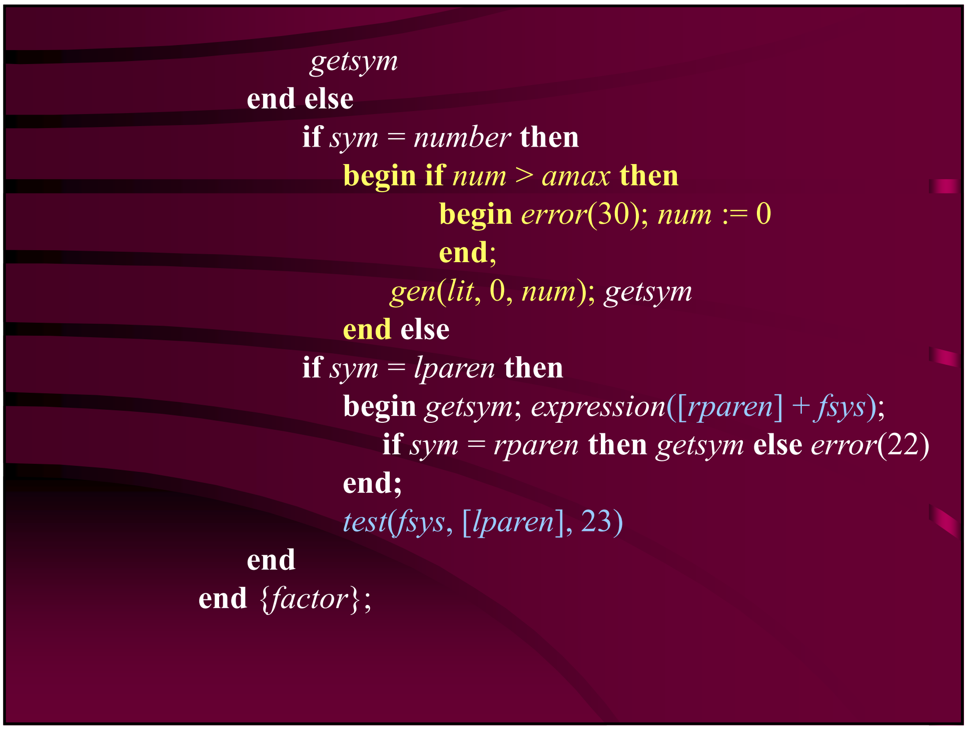
```
  function position(id: alfa): integer;
         var i: integer;
     begin {find identifier id in table}
         table[0].name := id; i := tx;
         while table[i].name ≠ id do i := i – 1;
          position := i
     end {position};
  procedure constdeclaration;
  begin if sym = ident then
         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};
```



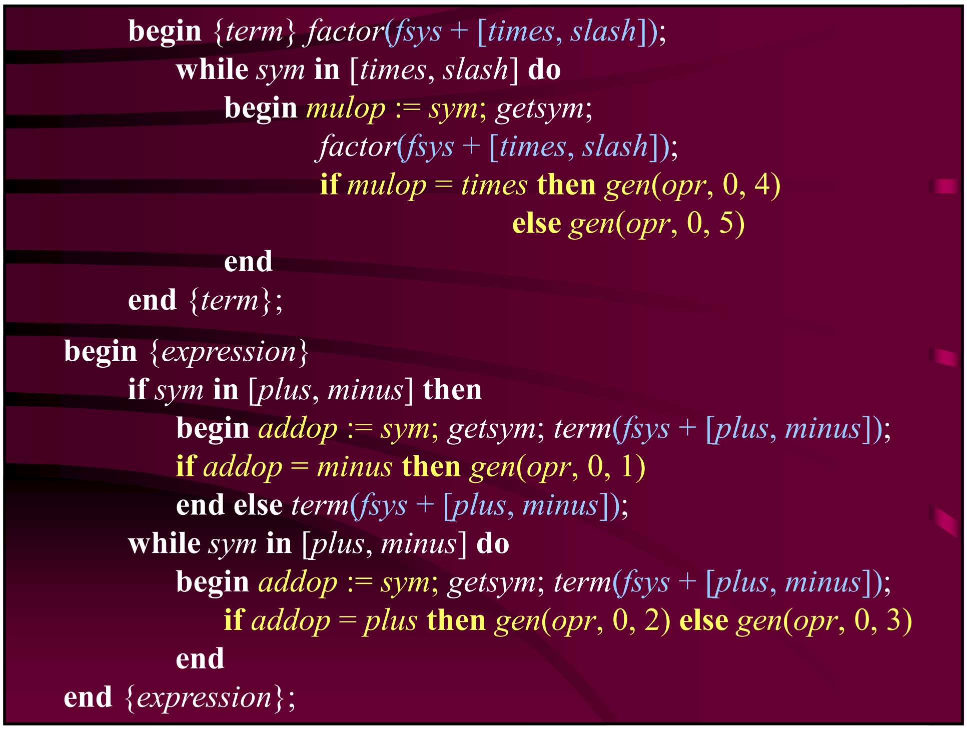
```
procedure vardeclaration;
begin if sym = ident then
        begin enter(variable); getsym
        end else error(4)
end {vardeclaration};


procedure listcode;
    var i: integer;
begin {list code generated for this block}
    for i := cx0 to cx − 1 do
        with code[i] do
            writeln(i, mnemonic[f]: 5, l: 3, a: 5)
end {listcode};
```



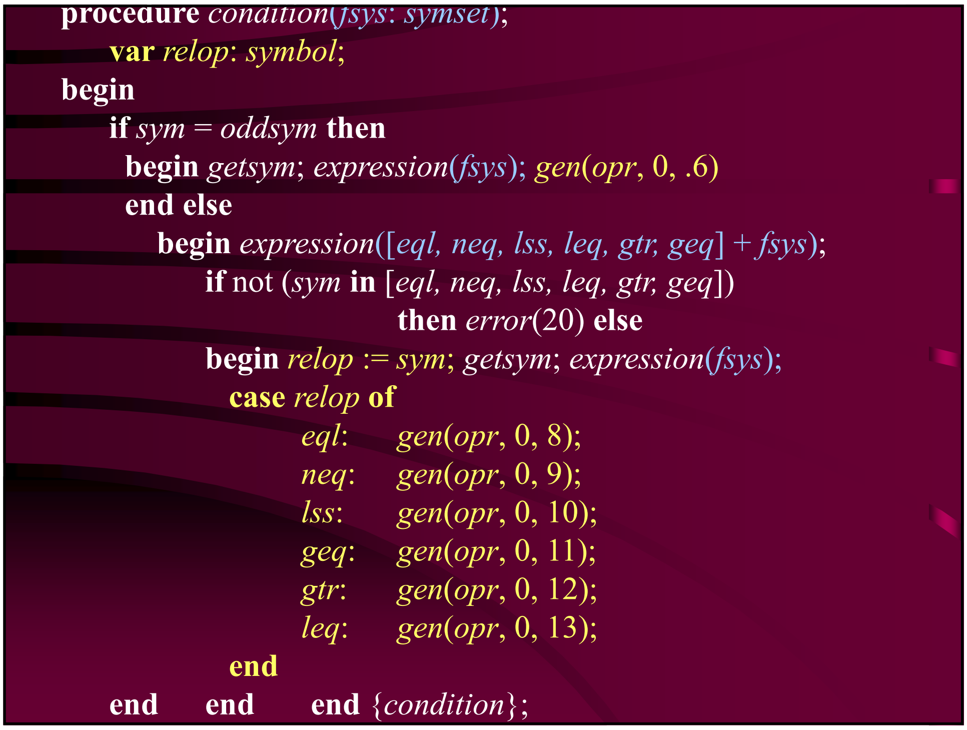
```
procedure statement (fsys: symset);
    var i, cx1, cx2 : integer;
    procedure expression (fsys: symset);
        var addop: symbol;
        procedure term (fsys: symset);
             var mulop: symbol;
             procedure factor (fsys: symset);
                 var i: integer;
            begin test(poczczyn, fpocz, 24);
                 while sym in poczczyn 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);
                                         procedure: error(21)
                                 end;
```



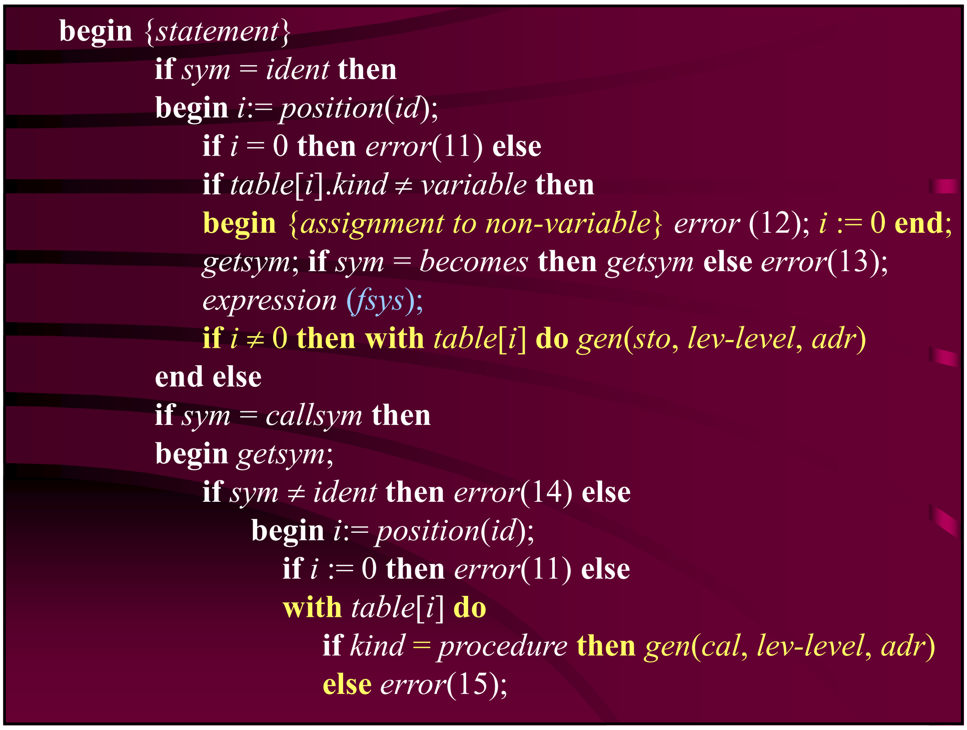
```
          getsym
    end else
        if sym = number then
            begin if num > amax then
                    begin error(30); num := 0
                    end;
                gen(lit, 0, num); getsym
            end else
        if sym = lparen then
            begin getsym; expression([rparen] + fsys);
               if sym = rparen then getsym else error(22)
            end;
            test(fsys, [lparen], 23)
    end
end {factor};
```



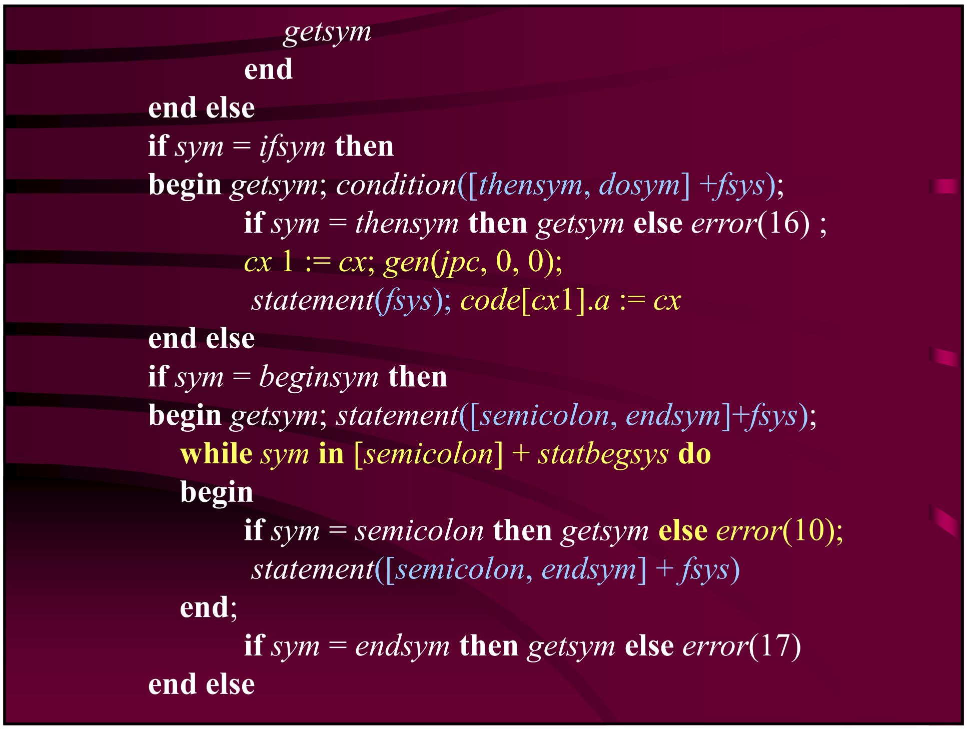
```
     begin {term} factor(fsys + [times, slash]);
         while sym in [times, slash] do
             begin mulop := sym; getsym;
                     factor(fsys + [times, slash]);
                     if mulop = times then gen(opr, 0, 4)
                                     else gen(opr, 0, 5)
             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};
```



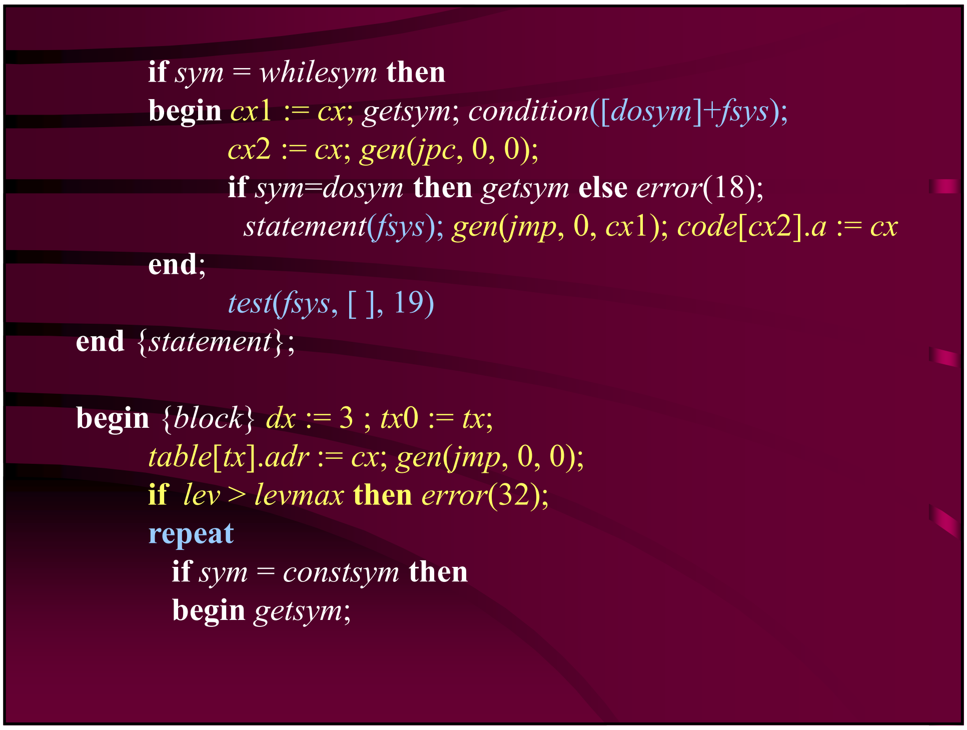
```
procedure condition(fsys: symset);
    var relop: symbol;
begin
    if sym = oddsym then
      begin getsym; expression(fsys); gen(opr, 0, .6)
      end else
         begin expression([eql, neq, lss, leq, gtr, geq] + fsys);
             if not (sym in [eql, neq, lss, leq, gtr, geq])
                             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};
```



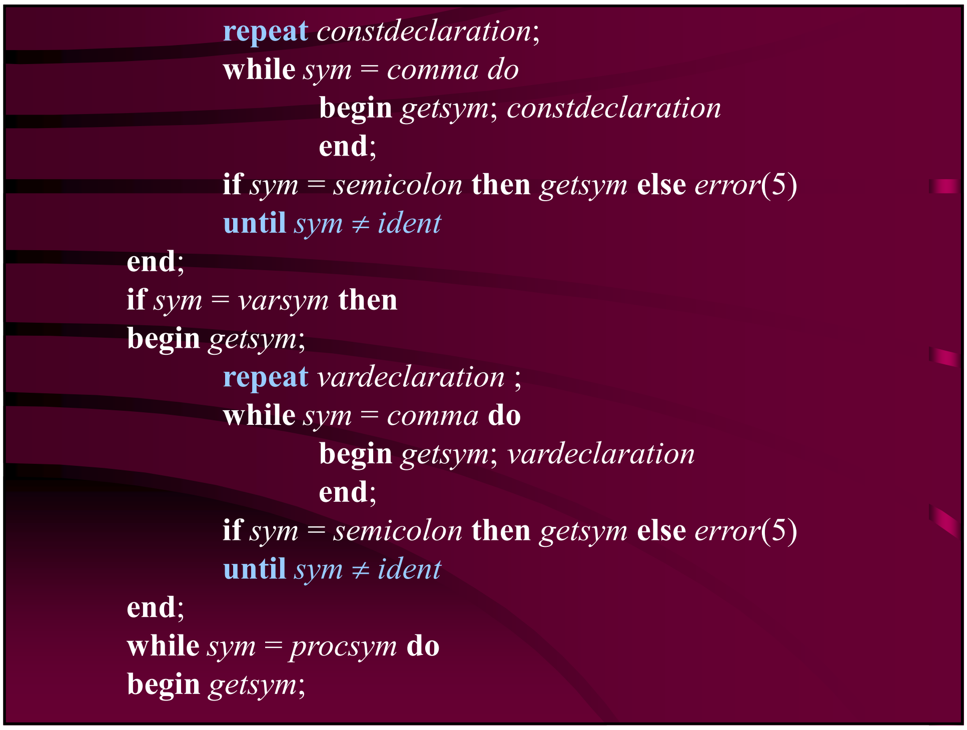
```
begin {statement}
        if sym = ident then
        begin i:= position(id);
            if i = 0 then error(11) else
            if table[i].kind ≠ variable then
            begin {assignment to non-variable} error (12); i := 0 end;
            getsym; if sym = becomes then getsym else error(13);
            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);
```



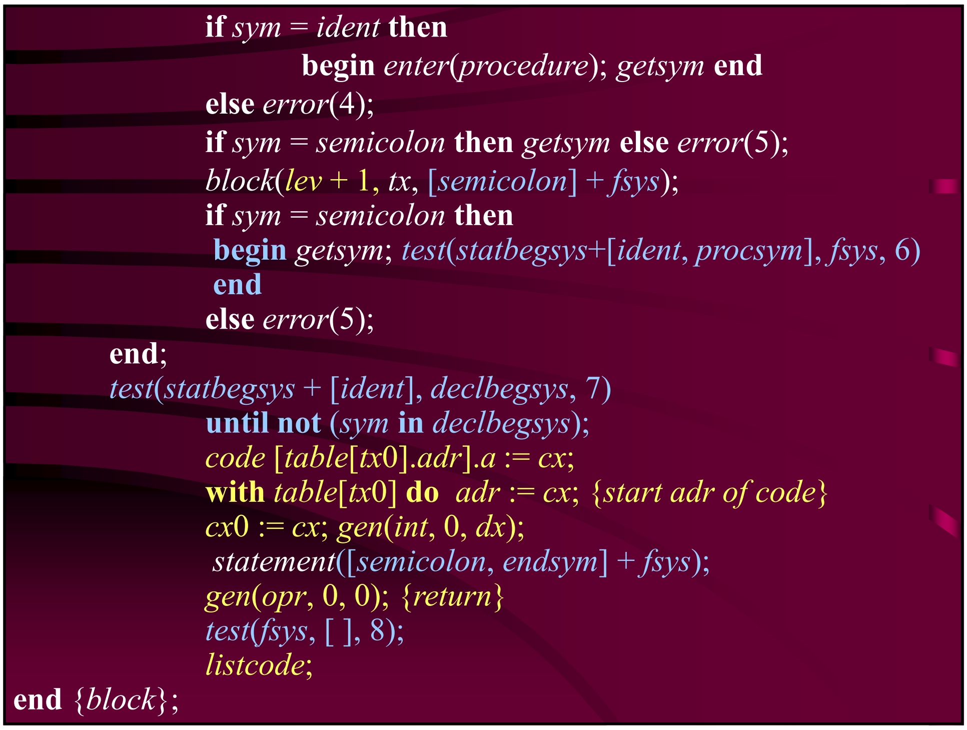
```
                    getsym
                 end
          end else
          if sym = ifsym then
          begin getsym; condition([thensym, dosym] +fsys);
                 if sym = thensym then getsym else error(16) ;
                 cx 1 := cx; gen(jpc, 0, 0);
                  statement(fsys); 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
```



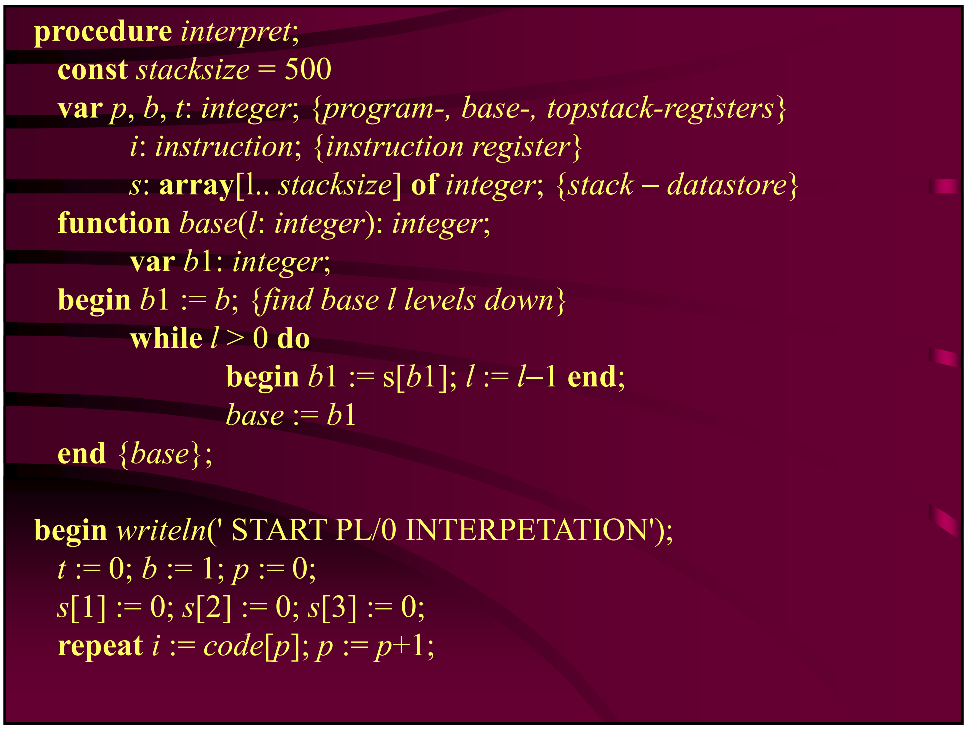
```
          if sym = whilesym then
          begin cx1 := cx; getsym; condition([dosym]+fsys);
                cx2 := cx; gen(jpc, 0, 0);
                if sym=dosym then getsym else error(18);
                 statement(fsys); gen(jmp, 0, cx1); code[cx2].a := cx
          end;
                test(fsys, [ ], 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;
```



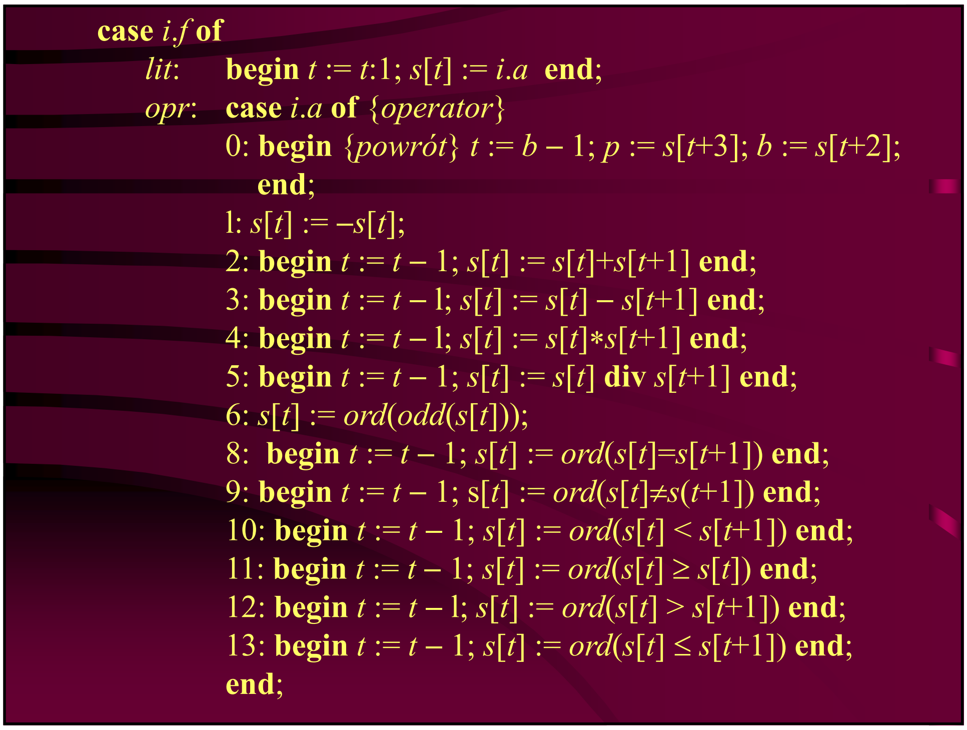
```
        repeat constdeclaration;
        while sym = comma do
            begin getsym; constdeclaration
            end;
        if sym = semicolon then getsym else error(5)
        until sym ≠ ident
    end;
    if sym = varsym then
    begin getsym;
        repeat vardeclaration ;
        while 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;
```



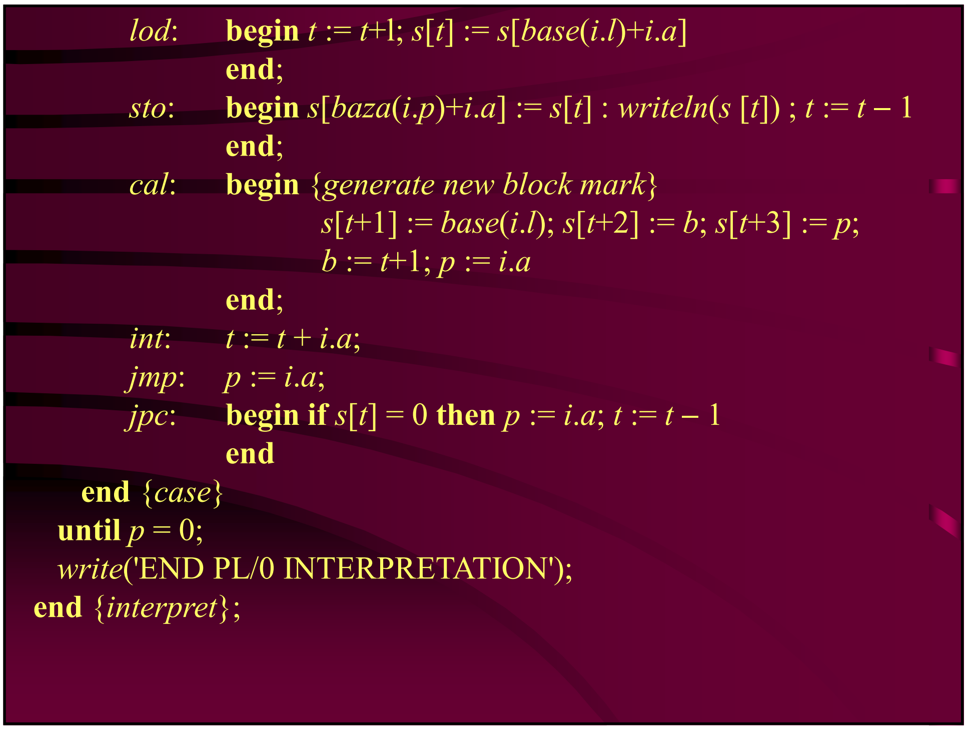
```
        if sym = ident then
            begin enter(procedure); getsym end
        else error(4);
        if sym = semicolon then getsym else error(5);
        block(lev + 1, tx, [semicolon] + fsys);
        if sym = semicolon then
         begin getsym; test(statbegsys+[ident, procsym], fsys, 6)
         end
        else error(5);
    end;
    test(statbegsys + [ident], declbegsys, 7)
        until not (sym in declbegsys);
        code [table[tx0].adr].a := cx;
        with table[tx0] do  adr := cx; {start adr of code}
        cx0 := cx; gen(int, 0, dx);
         statement([semicolon, endsym] + fsys);
        gen(opr, 0, 0); {return}
        test(fsys, [ ], 8);
        listcode;
end {block};
```



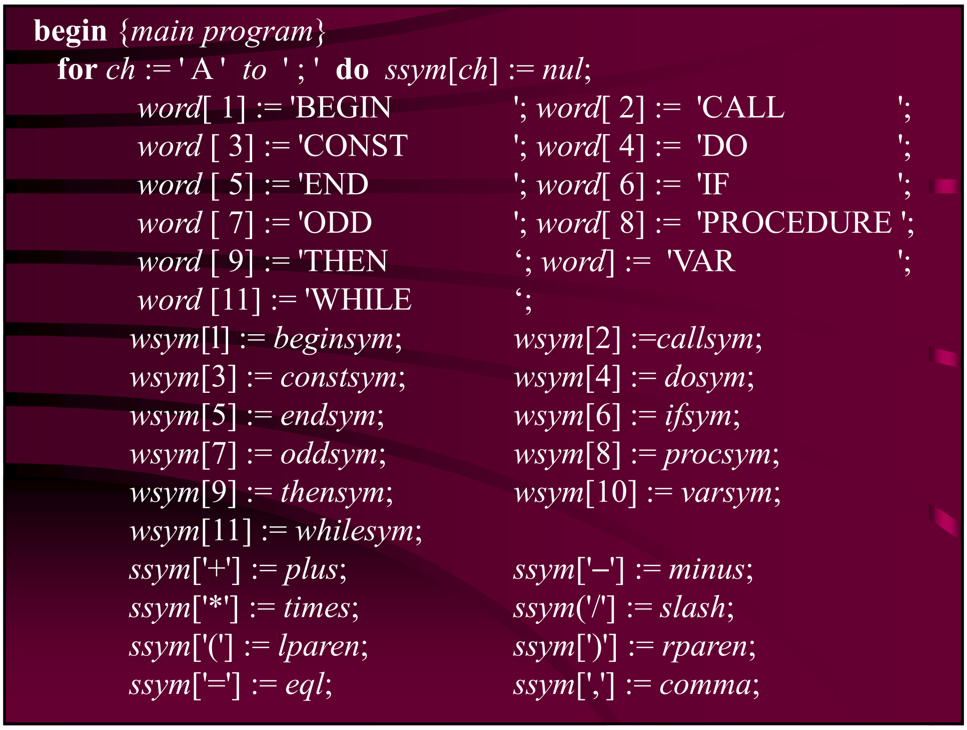
```
procedure interpret;
  const stacksize = 500
  var p, b, t: integer; {program-, base-, topstack-registers}
        i: instruction; {instruction register}
        s: array[1.. stacksize] of integer; {stack – datastore}
  function base(l: integer): integer;
        var b1: integer;
  begin b1 := b; {find base l levels down}
        while l > 0 do
                begin b1 := s[b1]; l := l–1 end;
                base := b1
  end {base};

begin writeln(' START PL/0 INTERPETATION');
  t := 0; b := 1; p := 0;
  s[1] := 0; s[2] := 0; s[3] := 0;
  repeat i := code[p]; p := p+1;
```



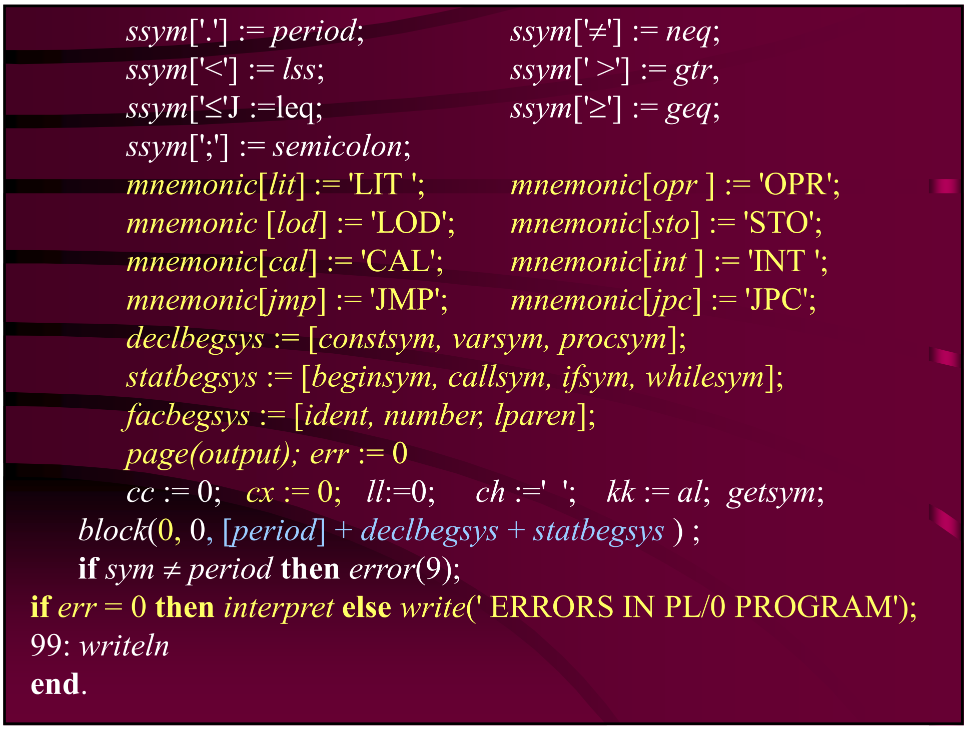
```
     case i.f of
        lit:    begin t := t:1; s[t] := i.a  end;
        opr:    case i.a of {operator}
                0: begin {powrót} t := b – 1; p := s[t+3]; b := s[t+2];
                   end;
                1: s[t] := –s[t];
                2: begin t := t – 1; s[t] := s[t]+s[t+1] end;
                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]) 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(i.l)+i.a]
                    end;
            sto:    begin s[baza(i.p)+i.a] := s[t] : writeln(s [t]) ; t := t – 1
                    end;
            cal:    begin {generate new block mark}
                            s[t+1] := base(i.l); s[t+2] := b; s[t+3] := p;
                            b := t+1; p := i.a
                    end;
            int:    t := t + i.a;
            jmp:    p := i.a;
            jpc:    begin if s[t] = 0 then p := i.a; t := t – 1
                    end
        end {case}
    until p = 0;
    write('END PL/0 INTERPRETATION');
end {interpret};
```



```
begin {main program}
  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] :=  'VAR           ';
          word [11] := 'WHILE         ‘;
          wsym[l] := 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['≤'J :=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.
```