```
program PASCALS(INPUT,OUTPUT,PRD,PRR);
{  author:N.Wirth, E.T.H. CH-8092 Zurich,1.3.76 }
{  modified by R.E.Berry
    Department of computer studies
    University of Lancaster

    Variants of this program are used on
    Data General Nova,Apple,and
    Western Digital Microengine machines. }
{   further modified by M.Z.Jin
    Department of Computer Science&Engineering BUAA,0ct.1989
}
const nkw = 27;    { no. of key words }
      alng = 10;   { no. of significant chars in identifiers }
      llng = 121;  { input line length }
      emax = 322;  { max exponent of real numbers }
      emin = -292; { min exponent }
      kmax = 15;   { max no. of significant digits }
      tmax = 100;  { size of table }
      bmax = 20;   { size of block-talbe }
      amax = 30;   { size of array-table }
      c2max = 20;  { size of real constant table }
      csmax = 30;  { max no. of cases }
      cmax = 800;  { size of code }
      lmax = 7;    { maximum level }
      smax = 600;  { size of string-table }
      ermax = 58;  { max error no. }
      omax = 63;   { highest order code }
      xmax = 32767;  { 2**15-1 }
      nmax = 32767;  { 2**15-1 }
      lineleng = 132; { output line length }
      linelimit = 200;
      stacksize = 1450;
type symbol = ( intcon, realcon, charcon, stringcon,
                notsy, plus, minus, times, idiv, rdiv, imod, andsy, orsy,
                eql, neq, gtr, geq, lss, leq,
                lparent, rparent, lbrack, rbrack, comma, semicolon, period,
                colon, becomes, constsy, typesy, varsy, funcsy,
                procsy, arraysy, recordsy, programsy, ident,
                beginsy, ifsy, casesy, repeatsy, whilesy, forsy,
                endsy, elsesy, untilsy, ofsy, dosy, tosy, downtosy, thensy);
     index = -xmax..+xmax;
     alfa = packed array[1..alng]of char;
     objecttyp = (konstant, vvariable, typel, prozedure, funktion );
     types = (notyp, ints, reals, bools, chars, arrays, records );
     symset = set of symbol;
     typset = set of types;
     item = record
               typ: types;
               ref: index;
            end;

     order = packed record
                f: -omax..+omax;
x: -lmax..+lmax;
```

```
                y: -nmax..+nmax
end;
var   ch:         char; { last character read from source program }
      rnum:       real; { real number from insymbol }
inum:       integer;     { integer from insymbol }
sleng:      integer;     { string length }
      cc:         integer;     { character counter }
      lc:         integer;     { program location counter }
      ll:         integer;     { length of current line }
      errpos:     integer;
      t,a,b,sx,c1,c2:integer;  { indices to tables }
      iflag, oflag, skipflag, stackdump, prtables: boolean;
      sy:         symbol;      { last symbol read by insymbol }
      errs:       set of 0..ermax;
      id:         alfa;        { identifier from insymbol }
      progname:   alfa;
      stantyps:   typset;
      constbegsys, typebegsys, blockbegsys, facbegsys, statbegsys: symset;
      line:       array[1..llng] of char;
      key:        array[1..nkw] of alfa;
      ksy:        array[1..nkw] of symbol;
      sps:        array[char]of symbol;  { special symbols }
      display:    array[0..lmax] of integer;
      tab:        array[0..tmax] of      { indentifier lable }
                  packed record
                      name: alfa;
                      link: index;
                      obj:  objecttyp;
                      typ:  types;
                      ref:  index;
normal: boolean;
                      lev:  0..lmax;
                      adr: integer
end;
     atab:       array[1..amax] of    { array-table }
                 packed record
                     inxtyp,eltyp: types;
                     elref,low,high,elsize,size: index
                 end;
     btab:       array[1..bmax] of    { block-table }
                 packed record
                     last, lastpar, psize, vsize: index
                 end;
     stab:       packed array[0..smax] of char; { string table }
     rconst:     array[1..c2max] of real;
     code:       array[0..cmax] of order;
     psin,psout,prr,prd:text;      { default in pascal p }
     inf, outf, fprr: string;

procedure errormsg;
  var k : integer;
     msg: array[0..ermax] of alfa;
  begin
    msg[0] := 'undef id  ';    msg[1] := 'multi def ';
    msg[2] := 'identifier';    msg[3] := 'program   ';
msg[4] := ')         ';    msg[5] := ':         ';
    msg[6] := 'syntax    ';    msg[7] := 'ident,var ';
msg[8] := 'of        ';    msg[9] := '(         ';
```

```
    msg[10] := 'id,array  ';    msg[11] := '(         ';
    msg[12] := ']         ';    msg[13] := '..        ';
    msg[14] := ';         ';    msg[15] := 'func. type';
    msg[16] := '=         ';    msg[17] := 'boolean   ';
    msg[18] := 'convar typ';    msg[19] := 'type      ';
    msg[20] := 'prog.param';    msg[21] := 'too big   ';
    msg[22] := '.         ';    msg[23] := 'type(case)';
    msg[24] := 'character ';    msg[25] := 'const id  ';
    msg[26] := 'index type';    msg[27] := 'indexbound';
    msg[28] := 'no array  ';    msg[29] := 'type id   ';
    msg[30] := 'undef type';    msg[31] := 'no record ';
    msg[32] := 'boole type';    msg[33] := 'arith type';
msg[34] := 'integer   ';    msg[35] := 'types     ';
    msg[36] := 'param type';    msg[37] := 'variab id ';
msg[38] := 'string    ';    msg[39] := 'no.of pars';
    msg[40] := 'real numbr';    msg[41] := 'type      ';
msg[42] := 'real type ';    msg[43] := 'integer   ';
    msg[44] := 'var,const ';    msg[45] := 'var,proc  ';
msg[46] := 'types(:=) ';    msg[47] := 'typ(case) ';
    msg[48] := 'type      ';    msg[49] := 'store ovfl';
    msg[50] := 'constant  ';    msg[51] := ':=        ';
    msg[52] := 'then      ';    msg[53] := 'until     ';
    msg[54] := 'do        ';    msg[55] := 'to downto ';
    msg[56] := 'begin     ';    msg[57] := 'end       ';
    msg[58] := 'factor';

    writeln(psout);
    writeln(psout,'key words');
    k := 0;
    while errs <> [] do
      begin
        while not( k in errs )do k := k + 1;
writeln(psout, k, ' ', msg[k] );
        errs := errs - [k]
end { while errs }
  end { errormsg } ;

procedure endskip;
  begin                 { underline skipped part of input }
    while errpos < cc do
      begin
        write( psout, '-');
        errpos := errpos + 1
      end;
    skipflag := false
  end { endskip };


procedure nextch;  { read next character; process line end }
  begin
    if cc = ll
    then begin
           if eof( psin )
           then begin
                  writeln( psout );
                  writeln( psout, 'program incomplete' );
                  errormsg;
                  exit;
```

```
                end;
            if errpos <> 0
            then begin
                   if skipflag then endskip;
                   writeln( psout );
                   errpos := 0
                end;
            write( psout, lc: 5, ' ');
            ll := 0;
            cc := 0;
            while not eoln( psin ) do
              begin
                ll := ll + 1;
                read( psin, ch );
                write( psout, ch );
                line[ll] := ch
              end;
            ll := ll + 1;
            readln( psin );
            line[ll] := ' ';
            writeln( psout );
          end;
          cc := cc + 1;
          ch := line[cc];
  end { nextch };

procedure error( n: integer );
begin
  if errpos = 0
  then write ( psout, '****' );
  if cc > errpos
  then begin
         write( psout, ' ': cc-errpos, '^', n:2);
errpos := cc + 3;
         errs := errs +[n]
end
end { error };

procedure fatal( n: integer );
  var msg : array[1..7] of alfa;
  begin
    writeln( psout );
    errormsg;
    msg[1] := 'identifier';   msg[2] := 'procedures';
    msg[3] := 'reals     ';   msg[4] := 'arrays    ';
msg[5] := 'levels    ';   msg[6] := 'code      ';
    msg[7] := 'strings   ';
writeln( psout, 'compiler table for ', msg[n], ' is too small');
    exit; {terminate compilation }
  end { fatal };

procedure insymbol;  {reads next symbol}
label 1,2,3;
  var  i,j,k,e: integer;
procedure readscale;
    var s,sign: integer;
    begin
      nextch;
```

```
      sign := 1;
      s := 0;
      if ch = '+'
      then nextch
      else if ch = '-'
           then begin
                  nextch;
                  sign := -1
                end;
      if not(( ch >= '0' )and (ch <= '9' ))
      then error( 40 )
      else repeat
           s := 10*s + ord( ord(ch)-ord('0'));
           nextch;
           until not(( ch >= '0' ) and ( ch <= '9' ));
      e := s*sign + e
    end { readscale };

  procedure adjustscale;
    var s : integer;
        d, t : real;
    begin
      if k + e > emax
      then error(21)
      else if k + e < emin
           then rnum := 0
           else begin
                  s := abs(e);
                  t := 1.0;
                  d := 10.0;
                  repeat
                    while not odd(s) do
                      begin
                        s := s div 2;
                        d := sqr(d)
                      end;
                    s := s - 1;
                    t := d * t
                  until s = 0;
if e >= 0
                  then rnum := rnum * t
                  else rnum := rnum / t
end
    end { adjustscale };

  procedure options;
    procedure switch( var b: boolean );
      begin
        b := ch = '+';
        if not b
        then if not( ch = '-' )
             then begin { print error message }
                    while( ch <> '*' ) and ( ch <> ',' ) do
                      nextch;
                  end
             else nextch
        else nextch
      end { switch };
```

```
    begin { options  }
      repeat
        nextch;
        if ch <> '*'
        then begin
               if ch = 't'
               then begin
                      nextch;
                      switch( prtables )
                    end
               else if ch = 's'
                  then begin
                         nextch;
                         switch( stackdump )
                       end;

             end
      until ch <> ','
    end { options };
  begin { insymbol  }
  1: while( ch = ' ' ) or ( ch = chr(9) ) do
       nextch;    { space & htab }
    case ch of
      'a','b','c','d','e','f','g','h','i',
      'j','k','l','m','n','o','p','q','r',
      's','t','u','v','w','x','y','z':
        begin { identifier of wordsymbol }
          k := 0;
          id := '          ';
          repeat
            if k < alng
            then begin
                   k := k + 1;
                   id[k] := ch
                 end;
            nextch
          until not((( ch >= 'a' ) and ( ch <= 'z' )) or (( ch >= '0') and (ch
<= '9' )));
          i := 1;
          j := nkw; { binary search }
          repeat
            k := ( i + j ) div 2;
            if id <= key[k]
            then j := k - 1;
            if id >= key[k]
            then i := k + 1;
          until i > j;
          if i - 1 > j
          then sy := ksy[k]
          else sy := ident
        end;
      '0','1','2','3','4','5','6','7','8','9':
        begin { number }
          k := 0;
          inum := 0;
          sy := intcon;
repeat
            inum := inum * 10 + ord(ch) - ord('0');
```

```
k := k + 1;
            nextch
          until not (( ch >= '0' ) and ( ch <= '9' ));
          if( k > kmax ) or ( inum > nmax )
          then begin
                 error(21);
                 inum := 0;
                 k := 0
               end;
          if ch = '.'
          then begin
                 nextch;
                 if ch = '.'
                 then ch := ':'
                 else begin
                        sy := realcon;
                        rnum := inum;
                        e := 0;
                        while ( ch >= '0' ) and ( ch <= '9' ) do
begin
                            e := e - 1;
                            rnum := 10.0 * rnum + (ord(ch) - ord('0'));
nextch
                          end;
                        if e = 0
                        then error(40);
                        if ch = 'e'
                        then readscale;
                        if e <> 0 then adjustscale
                      end
               end
          else if ch = 'e'
               then begin
sy := realcon;
                      rnum := inum;
                      e := 0;
readscale;
                      if e <> 0
                      then adjustscale
                    end;
        end;
      ':':
        begin
          nextch;
          if ch = '='
          then begin
                 sy := becomes;
                 nextch
               end
          else  sy := colon
         end;
      '<':
        begin
          nextch;
          if ch = '='
          then begin
                 sy := leq;
                 nextch
```

```
            end
        else
          if ch = '>'
          then begin
                 sy := neq;
                 nextch
               end
          else  sy := lss
      end;
    '>':
      begin
        nextch;
        if ch = '='
        then begin
               sy := geq;
               nextch
             end
        else  sy := gtr
      end;
    '.':
      begin
        nextch;
        if ch = '.'
        then begin
               sy := colon;
               nextch
             end
        else sy := period
      end;
    '''':
      begin
        k := 0;
2:      nextch;
        if ch = ''''
        then begin
               nextch;
               if ch <> ''''
               then goto 3
             end;
        if sx + k = smax
        then fatal(7);
        stab[sx+k] := ch;
        k := k + 1;
        if cc = 1
        then begin { end of line }
               k := 0;
             end
        else goto 2;
3:      if k = 1
        then begin
               sy := charcon;
               inum := ord( stab[sx] )
             end
        else if k = 0
             then begin
                    error(38);
                    sy := charcon;
                    inum := 0
```

```
                    end
                else begin
                       sy := stringcon;
                       inum := sx;
                       sleng := k;
                       sx := sx + k
                     end
        end;
      '(':
        begin
          nextch;
          if ch <> '*'
          then sy := lparent
          else begin { comment }
                 nextch;
                 if ch = '$'
                 then options;
                 repeat
                   while ch <> '*' do nextch;
                   nextch
                 until ch = ')';
                 nextch;
                 goto 1
               end
        end;
      '{':
        begin
          nextch;
          if ch = '$'
          then options;
          while ch <> '}' do
            nextch;
          nextch;
          goto 1
        end;
      '+', '-', '*', '/', ')', '=', ',', '[', ']', ';':
        begin
          sy := sps[ch];
          nextch
        end;
      '$','"' ,'@', '?', '&', '^', '!':
        begin
          error(24);
          nextch;
          goto 1
        end
      end { case }
    end { insymbol };

procedure enter(x0:alfa; x1:objecttyp; x2:types; x3:integer );
  begin
    t := t + 1;    { enter standard identifier }
    with tab[t] do
      begin
        name := x0;
        link := t - 1;
        obj := x1;
        typ := x2;
```

```
        ref := 0;
        normal := true;
        lev := 0;
        adr := x3;
      end
  end; { enter }

procedure enterarray( tp: types; l,h: integer );
  begin
    if l > h
    then error(27);
    if( abs(l) > xmax ) or ( abs(h) > xmax )
    then begin
           error(27);
           l := 0;
           h := 0;
         end;
    if a = amax
    then fatal(4)
    else begin
           a := a + 1;
           with atab[a] do
             begin
               inxtyp := tp;
               low := l;
               high := h
             end
         end
  end { enterarray };

procedure enterblock;
  begin
    if b = bmax
    then fatal(2)
    else begin
           b := b + 1;
           btab[b].last := 0;
           btab[b].lastpar := 0;
         end
  end { enterblock };

procedure enterreal( x: real );
  begin
    if c2 = c2max - 1
    then fatal(3)
    else begin
           rconst[c2+1] := x;
           c1 := 1;
           while rconst[c1] <> x do
             c1 := c1 + 1;
           if c1 > c2
           then  c2 := c1
         end
  end { enterreal };

procedure emit( fct: integer );
  begin
    if lc = cmax
```

```
    then fatal(6);
code[lc].f := fct;
    lc := lc + 1
end { emit };


procedure emit1( fct, b: integer );
  begin
    if lc = cmax
    then fatal(6);
    with code[lc] do
      begin
        f := fct;
        y := b;
      end;
    lc := lc + 1
  end { emit1 };

procedure emit2( fct, a, b: integer );
  begin
    if lc = cmax then fatal(6);
    with code[lc] do
      begin
        f := fct;
        x := a;
        y := b
      end;
    lc := lc + 1;
end { emit2 };

procedure printtables;
  var  i: integer;
o: order;
      mne: array[0..omax] of
           packed array[1..5] of char;
  begin
    mne[0] := 'LDA  ';   mne[1] := 'LOD  ';  mne[2] := 'LDI  ';
mne[3] := 'DIS  ';   mne[8] := 'FCT  ';  mne[9] := 'INT  ';
    mne[10] := 'JMP  ';   mne[11] := 'JPC  ';  mne[12] := 'SWT  ';
    mne[13] := 'CAS  ';   mne[14] := 'F1U  ';  mne[15] := 'F2U  ';
    mne[16] := 'F1D  ';   mne[17] := 'F2D  ';  mne[18] := 'MKS  ';
    mne[19] := 'CAL  ';   mne[20] := 'IDX  ';  mne[21] := 'IXX  ';
    mne[22] := 'LDB  ';   mne[23] := 'CPB  ';  mne[24] := 'LDC  ';
mne[25] := 'LDR  ';   mne[26] := 'FLT  ';  mne[27] := 'RED  ';
mne[28] := 'WRS  ';   mne[29] := 'WRW  ';  mne[30] := 'WRU  ';
    mne[31] := 'HLT  ';   mne[32] := 'EXP  ';  mne[33] := 'EXF  ';
    mne[34] := 'LDT  ';   mne[35] := 'NOT  ';  mne[36] := 'MUS  ';
mne[37] := 'WRR  ';   mne[38] := 'STO  ';  mne[39] := 'EQR  ';
mne[40] := 'NER  ';   mne[41] := 'LSR  ';  mne[42] := 'LER  ';
    mne[43] := 'GTR  ';   mne[44] := 'GER  ';  mne[45] := 'EQL  ';
mne[46] := 'NEQ  ';   mne[47] := 'LSS  ';  mne[48] := 'LEQ  ';
    mne[49] := 'GRT  ';   mne[50] := 'GEQ  ';  mne[51] := 'ORR  ';
    mne[52] := 'ADD  ';   mne[53] := 'SUB  ';  mne[54] := 'ADR  ';
    mne[55] := 'SUR  ';   mne[56] := 'AND  ';  mne[57] := 'MUL  ';
    mne[58] := 'DIV  ';   mne[59] := 'MOD  ';  mne[60] := 'MUR  ';
    mne[61] := 'DIR  ';   mne[62] := 'RDL  ';  mne[63] := 'WRL  ';

writeln(psout);
```

```
    writeln(psout);
    writeln(psout);
    writeln(psout,'   identifiers  link  obj  typ  ref  nrm  lev  adr');
    writeln(psout);
    for i := btab[1].last to t do
      with tab[i] do
        writeln( psout, i,' ', name, link:5, ord(obj):5, ord(typ):5,ref:5,
ord(normal):5,lev:5,adr:5);
    writeln( psout );
    writeln( psout );
    writeln( psout );
    writeln( psout, 'blocks   last  lpar  psze  vsze' );
    writeln( psout );
    for i := 1 to b do
       with btab[i] do
         writeln( psout, i:4, last:9, lastpar:5, psize:5, vsize:5 );
    writeln( psout );
    writeln( psout );
    writeln( psout );
    writeln( psout, 'arrays xtyp etyp eref low high elsz size');
    writeln( psout );
    for i := 1 to a do
      with atab[i] do
        writeln( psout, i:4, ord(inxtyp):9, ord(eltyp):5, elref:5, low:5,
high:5, elsize:5, size:5);
    writeln( psout );
    writeln( psout );
    writeln( psout );
    writeln( psout, 'code:');
    writeln( psout );
    for i := 0 to lc-1 do
      begin
write( psout, i:5 );
        o := code[i];
write( psout, mne[o.f]:8, o.f:5 );
        if o.f < 31
        then if o.f < 4
             then write( psout, o.x:5, o.y:5 )
             else write( psout, o.y:10 )
        else write( psout, '          ' );
        writeln( psout, ',' )
      end;
    writeln( psout );
    writeln( psout, 'Starting address is ', tab[btab[1].last].adr:5 )
  end { printtables };


procedure block( fsys: symset; isfun: boolean; level: integer );
  type conrec = record
                  case tp: types of
                    ints, chars, bools : ( i:integer );
                    reals :( r:real )
                end;
  var dx : integer ;  { data allocation index }
      prt: integer ;  { t-index of this procedure }
      prb: integer ;  { b-index of this procedure }
x  : integer ;
```

```
  procedure skip( fsys:symset; n:integer);
begin
      error(n);
      skipflag := true;
      while not ( sy in fsys ) do
        insymbol;
      if skipflag then endskip
    end { skip };

  procedure test( s1,s2: symset; n:integer );
    begin
      if not( sy in s1 )
      then skip( s1 + s2, n )
    end { test };

  procedure testsemicolon;
    begin
      if sy = semicolon
      then insymbol
      else begin
             error(14);
             if sy in [comma, colon]
             then insymbol
           end;
      test( [ident] + blockbegsys, fsys, 6 )
    end { testsemicolon };


  procedure enter( id: alfa; k:objecttyp );
    var j,l : integer;
    begin
      if t = tmax
      then fatal(1)
      else begin
             tab[0].name := id;
             j := btab[display[level]].last;
             l := j;
             while tab[j].name <> id do
               j := tab[j].link;
             if j <> 0
             then error(1)
             else begin
                    t := t + 1;
                    with tab[t] do
                      begin
                        name := id;
                        link := l;
                        obj := k;
                        typ := notyp;
                        ref := 0;
                        lev := level;
                        adr := 0;
                        normal := false { initial value }
                      end;
                    btab[display[level]].last := t
                  end
           end
    end { enter };
```

```
  function loc( id: alfa ):integer;
    var i,j : integer;        { locate if in table }
    begin
      i := level;
      tab[0].name := id;  { sentinel }
      repeat
        j := btab[display[i]].last;
        while tab[j].name <> id do
j := tab[j].link;
        i := i - 1;
until ( i < 0 ) or ( j <> 0 );
      if j = 0
      then error(0);
      loc := j
    end { loc } ;

  procedure entervariable;
    begin
      if sy = ident
      then begin
             enter( id, vvariable );
             insymbol
           end
      else error(2)
    end { entervariable };

  procedure constant( fsys: symset; var c: conrec );
var x, sign : integer;
    begin
      c.tp := notyp;
c.i := 0;
      test( constbegsys, fsys, 50 );
      if sy in constbegsys
      then begin
             if sy = charcon
             then begin
                    c.tp := chars;
                    c.i := inum;
                    insymbol
                  end
             else begin
                  sign := 1;
                  if sy in [plus, minus]
                  then begin
                         if sy = minus
                         then sign := -1;
                         insymbol
                       end;
                  if sy = ident
                  then begin
                         x := loc(id);
                         if x <> 0
                         then
                           if tab[x].obj <> konstant
                           then error(25)
                           else begin
                                  c.tp := tab[x].typ;
```

```
                                if c.tp = reals
                                then c.r := sign*rconst[tab[x].adr]
                                else c.i := sign*tab[x].adr
                              end;
                        insymbol
                      end
                 else if sy = intcon
                      then begin
                             c.tp := ints;
                             c.i := sign*inum;
                             insymbol
                           end
                      else if sy = realcon
                           then begin
                                  c.tp := reals;
                                  c.r := sign*rnum;
                                  insymbol
                                end
                           else skip(fsys,50)
               end;
               test(fsys,[],6)
          end
    end { constant };

procedure typ( fsys: symset; var tp: types; var rf,sz:integer );
    var eltp : types;
        elrf, x : integer;
elsz, offset, t0, t1 : integer;

procedure arraytyp( var aref, arsz: integer );
var eltp : types;
      low, high : conrec;
      elrf, elsz: integer;
      begin
        constant( [colon, rbrack, rparent, ofsy] + fsys, low );
        if low.tp = reals
        then begin
               error(27);
               low.tp := ints;
               low.i := 0
             end;
        if sy = colon
        then insymbol
        else error(13);
        constant( [rbrack, comma, rparent, ofsy ] + fsys, high );
        if high.tp <> low.tp
        then begin
               error(27);
               high.i := low.i
             end;
        enterarray( low.tp, low.i, high.i );
        aref := a;
        if sy = comma
        then begin
               insymbol;
               eltp := arrays;
               arraytyp( elrf, elsz )
             end
```

```
        else begin
               if sy = rbrack
               then insymbol
               else begin
                      error(12);
                      if sy = rparent
                      then insymbol
                    end;
               if sy = ofsy
               then insymbol
               else error(8);
               typ( fsys, eltp, elrf, elsz )
             end;
             with atab[aref] do
               begin
                 arsz := (high-low+1) * elsz;
                 size := arsz;
                 eltyp := eltp;
                 elref := elrf;
                 elsize := elsz
               end
      end { arraytyp };
    begin { typ  }
      tp := notyp;
      rf := 0;
      sz := 0;
      test( typebegsys, fsys, 10 );
      if sy in typebegsys
      then begin
             if sy = ident
             then begin
                    x := loc(id);
                    if x <> 0
                    then with tab[x] do
                           if obj <> typel
                           then error(29)
                           else begin
                                  tp := typ;
                                  rf := ref;
                                  sz := adr;
                                  if tp = notyp
                                  then error(30)
                                end;
                    insymbol
                  end
             else if sy = arraysy
                  then begin
                         insymbol;
                         if sy = lbrack
                         then insymbol
                         else begin
                                error(11);
                                if sy = lparent
                                then insymbol
                              end;
                         tp := arrays;
                         arraytyp(rf,sz)
                         end
```

```
              else begin { records }
                     insymbol;
                     enterblock;
                     tp := records;
                     rf := b;
                     if level = lmax
                     then fatal(5);
                     level := level + 1;
                     display[level] := b;
                     offset := 0;
                     while not ( sy in fsys - [semicolon,comma,ident]+ [endsy] )
do
                       begin { field section }
                         if sy = ident
                         then begin
                                t0 := t;
                                entervariable;
                                while sy = comma do
                                  begin
                                    insymbol;
                                    entervariable
                                  end;
                                if sy = colon
                                then insymbol
                                else error(5);
                                t1 := t;
                                typ( fsys + [semicolon, endsy, comma,ident],
eltp, elrf, elsz );
                                while t0 < t1 do
                                begin
                                  t0 := t0 + 1;
                                  with tab[t0] do
                                    begin
                                      typ := eltp;
                                      ref := elrf;
                                      normal := true;
                                      adr := offset;
                                      offset := offset + elsz
                                    end
                                end
                              end; { sy = ident }
                         if sy <> endsy
                         then begin
                                if sy = semicolon
                                then insymbol
                                else begin
                                       error(14);
                                       if sy = comma
                                       then insymbol
                                     end;
                                     test( [ident,endsy, semicolon],fsys,6 )
                              end
                       end; { field section }
                     btab[rf].vsize := offset;
                     sz := offset;
                     btab[rf].psize := 0;
                     insymbol;
                     level := level - 1
```

```
                end; { record }
            test( fsys, [],6 )
          end;
      end { typ };

  procedure parameterlist; { formal parameter list  }
    var tp : types;
        valpar : boolean;
        rf, sz, x, t0 : integer;
    begin
      insymbol;
      tp := notyp;
      rf := 0;
      sz := 0;
      test( [ident, varsy], fsys+[rparent], 7 );
      while sy in [ident, varsy] do
        begin
          if sy <> varsy
          then valpar := true
          else begin
                 insymbol;
                 valpar := false
               end;
          t0 := t;
          entervariable;
          while sy = comma do
            begin
              insymbol;
              entervariable;
            end;
          if sy = colon
          then begin
                 insymbol;
                 if sy <> ident
                 then error(2)
                 else begin
                        x := loc(id);
                        insymbol;
                        if x <> 0
                        then with tab[x] do
                          if obj <> typel
                          then error(29)
                          else begin
                                 tp := typ;
                                 rf := ref;
                                 if valpar
                                 then sz := adr
                                 else sz := 1
                               end;
                      end;
                 test( [semicolon, rparent], [comma,ident]+fsys, 14 )
                 end
          else error(5);
          while t0 < t do
            begin
              t0 := t0 + 1;
              with tab[t0] do
                begin
```

```
                  typ := tp;
                  ref := rf;
adr := dx;
                  lev := level;
                  normal := valpar;
dx := dx + sz
                end
            end;
            if sy <> rparent
            then begin
                   if sy = semicolon
                   then insymbol
                   else begin
                          error(14);
                          if sy = comma
                          then insymbol
                        end;
                        test( [ident, varsy],[rparent]+fsys,6)
                end
        end { while };
      if sy = rparent
      then begin
             insymbol;
             test( [semicolon, colon],fsys,6 )
           end
      else error(4)
    end { parameterlist };


procedure constdec;
    var c : conrec;
begin
      insymbol;
      test([ident], blockbegsys, 2 );
      while sy = ident do
        begin
          enter(id, konstant);
          insymbol;
          if sy = eql
          then insymbol
          else begin
                 error(16);
                 if sy = becomes
                 then insymbol
               end;
          constant([semicolon,comma,ident]+fsys,c);
          tab[t].typ := c.tp;
          tab[t].ref := 0;
          if c.tp = reals
          then begin
enterreal(c.r);
                 tab[t].adr := c1;
end
          else tab[t].adr := c.i;
          testsemicolon
        end
    end { constdec };
```

```
  procedure typedeclaration;
var tp: types;
        rf, sz, t1 : integer;
begin
      insymbol;
      test([ident], blockbegsys,2 );
      while sy = ident do
        begin
          enter(id, typel);
          t1 := t;
          insymbol;
          if sy = eql
          then insymbol
          else begin
                 error(16);
                 if sy = becomes
                 then insymbol
               end;
          typ( [semicolon,comma,ident]+fsys, tp,rf,sz );
          with tab[t1] do
            begin
              typ := tp;
              ref := rf;
              adr := sz
            end;
          testsemicolon
        end
    end { typedeclaration };

  procedure variabledeclaration;
var tp : types;
        t0, t1, rf, sz : integer;
    begin
      insymbol;
while sy = ident do
        begin
          t0 := t;
          entervariable;
          while sy = comma do
            begin
              insymbol;
              entervariable;
            end;
          if sy = colon
          then insymbol
          else error(5);
          t1 := t;
          typ([semicolon,comma,ident]+fsys, tp,rf,sz );
          while t0 < t1 do
            begin
              t0 := t0 + 1;
              with tab[t0] do
                begin
                  typ := tp;
                  ref := rf;
                  lev := level;
                  adr := dx;
                  normal := true;
```

```
                dx := dx + sz
              end
            end;
          testsemicolon
        end
    end { variabledeclaration };

  procedure procdeclaration;
    var isfun : boolean;
    begin
      isfun := sy = funcsy;
      insymbol;
      if sy <> ident
      then begin
             error(2);
             id :='          '
           end;
      if isfun
      then enter(id,funktion)
      else enter(id,prozedure);
      tab[t].normal := true;
      insymbol;
      block([semicolon]+fsys, isfun, level+1 );
      if sy = semicolon
      then insymbol
      else error(14);
      emit(32+ord(isfun)) {exit}
    end { proceduredeclaration };


procedure statement( fsys:symset );
    var i : integer;

procedure expression(fsys:symset; var x:item); forward;
    procedure selector(fsys:symset; var v:item);
    var x : item;
        a,j : integer;
    begin { sy in [lparent, lbrack, period] }
      repeat
        if sy = period
        then begin
               insymbol; { field selector }
               if sy <> ident
               then error(2)
               else begin
                      if v.typ <> records
                      then error(31)
                      else begin { search field identifier }
                             j := btab[v.ref].last;
                             tab[0].name := id;
                             while tab[j].name <> id do
                               j := tab[j].link;
                             if j = 0
                             then error(0);
                             v.typ := tab[j].typ;
                             v.ref := tab[j].ref;
                             a := tab[j].adr;
                             if a <> 0
```

```
                          then emit1(9,a)
                        end;
                    insymbol
                  end
            end
        else begin { array selector }
               if sy <> lbrack
               then error(11);
               repeat
                 insymbol;
                 expression( fsys+[comma,rbrack],x);
                 if v.typ <> arrays
                 then error(28)
                 else begin
                        a := v.ref;
                        if atab[a].inxtyp <> x.typ
                        then error(26)
                        else if atab[a].elsize = 1
                             then emit1(20,a)
                             else emit1(21,a);
                        v.typ := atab[a].eltyp;
                        v.ref := atab[a].elref
                      end
               until sy <> comma;
               if sy = rbrack
               then insymbol
               else begin
                      error(12);
                      if sy = rparent
                      then insymbol
                    end
            end
      until not( sy in[lbrack, lparent, period]);
      test( fsys,[],6)
    end { selector };

    procedure call( fsys: symset; i:integer );
var x : item;
          lastp,cp,k : integer;
begin
        emit1(18,i); { mark stack }
        lastp := btab[tab[i].ref].lastpar;
        cp := i;
        if sy = lparent
        then begin { actual parameter list }
               repeat
                 insymbol;
                 if cp >= lastp
                 then error(39)
                 else begin
                        cp := cp + 1;
                        if tab[cp].normal
                        then begin { value parameter }
                               expression( fsys+[comma, colon,rparent],x);
                               if x.typ = tab[cp].typ
                               then begin
                                      if x.ref <> tab[cp].ref
                                      then error(36)
```

```
                                        else if x.typ = arrays
                                             then emit1(22,atab[x.ref].size)
                                             else if x.typ = records
                                                  then emit1(22,btab[x.ref].vsize)
                                   end
                              else if ( x.typ = ints ) and ( tab[cp].typ =
reals )
                                   then emit1(26,0)
                                   else if x.typ <> notyp
                                        then error(36);
                             end
                        else begin { variable parameter }
                               if sy <> ident
                               then error(2)
                               else begin
                                      k := loc(id);
                                      insymbol;
                                      if k <> 0
                                      then begin
                                             if tab[k].obj <> vvariable
                                             then error(37);
                                             x.typ := tab[k].typ;
                                             x.ref := tab[k].ref;
                                             if tab[k].normal
                                             then emit2(0,tab[k].lev,tab[k].adr)
else emit2(1,tab[k].lev,tab[k].adr);
if sy in [lbrack, lparent, period]
                                             then selector(fsys+
[comma,colon,rparent],x);
                                             if ( x.typ <> tab[cp].typ ) or (
x.ref <> tab[cp].ref )
                                             then error(36)
                                           end
                                   end
                             end {variable parameter }
                      end;
                  test( [comma, rparent],fsys,6)
              until sy <> comma;
              if sy = rparent
              then insymbol
              else error(4)
            end;
        if cp < lastp
        then error(39); { too few actual parameters }
        emit1(19,btab[tab[i].ref].psize-1 );
        if tab[i].lev < level
        then emit2(3,tab[i].lev, level )
      end { call };

    function resulttype( a, b : types) :types;
      begin
        if ( a > reals ) or ( b > reals )
        then begin
               error(33);
               resulttype := notyp
             end
        else if ( a = notyp ) or ( b = notyp )
             then resulttype := notyp
```

```
             else if a = ints
                  then if b = ints
                       then resulttype := ints
                       else begin
                              resulttype := reals;
                              emit1(26,1)
                            end
                  else begin
                         resulttype := reals;
                         if b = ints
                         then emit1(26,0)
                       end
      end { resulttype } ;

    procedure expression( fsys: symset; var x: item );
var y : item;
          op : symbol;

      procedure simpleexpression( fsys: symset; var x: item );
        var y : item;
            op : symbol;

        procedure term( fsys: symset; var x: item );
          var y : item;
              op : symbol;

          procedure factor( fsys: symset; var x: item );
            var i,f : integer;

            procedure standfct( n: integer );
              var ts : typset;
begin  { standard function no. n }
                if sy = lparent
                then insymbol
                else error(9);
                if n < 17
                then begin
                       expression( fsys+[rparent], x );
                       case n of
                       { abs, sqr } 0,2: begin
                                           ts := [ints, reals];
                                           tab[i].typ := x.typ;
                                           if x.typ = reals
                                           then n := n + 1
                                         end;
                       { odd, chr } 4,5: ts := [ints];
                       { odr }        6: ts := [ints,bools,chars];
                       { succ,pred } 7,8 : begin
                                             ts := [ints, bools,chars];
                                             tab[i].typ := x.typ
                                           end;
                       { round,trunc } 9,10,11,12,13,14,15,16:
                       { sin,cos,... }     begin
                                             ts := [ints,reals];
                                             if x.typ = ints
                                             then emit1(26,0)
                                           end;
                     end; { case }
```

```
                    if x.typ in ts
                    then emit1(8,n)
                    else if x.typ <> notyp
                         then error(48);
                  end
               else begin { n in [17,18] }
                      if sy <> ident
                      then error(2)
                      else if id <> 'input    '
                           then error(0)
                           else insymbol;
                      emit1(8,n);
                    end;
x.typ := tab[i].typ;
if sy = rparent
               then insymbol
               else error(4)
             end { standfct } ;
           begin { factor }
             x.typ := notyp;
             x.ref := 0;
             test( facbegsys, fsys,58 );
             while sy in facbegsys do
               begin
                 if sy = ident
                 then begin
                        i := loc(id);
                        insymbol;
                        with tab[i] do
                          case obj of
                            konstant: begin
                                        x.typ := typ;
                                        x.ref := 0;
                                        if x.typ = reals
then emit1(25,adr)
                                        else emit1(24,adr)
end;
                            vvariable:begin
                                        x.typ := typ;
                                        x.ref := ref;
                                        if sy in [lbrack, lparent,period]
                                        then begin
                                               if normal
                                               then f := 0
else f := 1;
                                               emit2(f,lev,adr);
selector(fsys,x);
                                               if x.typ in stantyps
                                               then emit(34)
                                             end
                                        else begin
                                               if x.typ in stantyps
                                               then if normal
                                                    then f := 1
                                                    else f := 2
                                               else if normal
                                                    then f := 0
else f := 1;
```

```
                                                emit2(f,lev,adr)
end
                                          end;
                                typel,prozedure: error(44);
                                funktion: begin
                                            x.typ := typ;
                                            if lev <> 0
                                            then call(fsys,i)
                                            else standfct(adr)
                                          end
                            end { case,with }
                       end
                  else if sy in [ charcon,intcon,realcon ]
                       then begin
                              if sy = realcon
                              then begin
x.typ := reals;
                                     enterreal(rnum);

emit1(25,c1)
                                   end
                              else begin
                                     if sy = charcon
                                     then x.typ := chars
                                     else x.typ := ints;
                                     emit1(24,inum)
                                   end;
                              x.ref := 0;
                              insymbol
                            end
                       else if sy = lparent
                            then begin
                                   insymbol;
                                   expression(fsys + [rparent],x);
                                   if sy = rparent
                                   then insymbol
                                   else error(4)
                                 end
                            else if sy = notsy
                                 then begin
                                        insymbol;
                                        factor(fsys,x);
                                        if x.typ = bools
                                        then emit(35)
                                        else if x.typ <> notyp
                                             then error(32)
                                      end;
                  test(fsys,facbegsys,6)
                end { while }
            end { factor };
          begin { term   }
            factor( fsys + [times,rdiv,idiv,imod,andsy],x);
            while sy in [times,rdiv,idiv,imod,andsy] do
              begin
                op := sy;
                insymbol;
                factor(fsys+[times,rdiv,idiv,imod,andsy],y );
                if op = times
                then begin
```

```
                    x.typ := resulttype(x.typ, y.typ);
                    case x.typ of
                      notyp: ;
                      ints : emit(57);
                      reals: emit(60);
                    end
                  end
               else if op = rdiv
                    then begin
                           if x.typ = ints
                           then begin
                                  emit1(26,1);
                                  x.typ := reals;
                                end;
                           if y.typ = ints
                           then begin
                                  emit1(26,0);
                                  y.typ := reals;
                                end;
                           if (x.typ = reals) and (y.typ = reals)
                           then emit(61)
                           else begin
                                  if( x.typ <> notyp ) and (y.typ <> notyp)
                                  then error(33);
                                  x.typ := notyp
                                end
                         end
                    else if op = andsy
                         then begin
                                if( x.typ = bools )and(y.typ = bools)
                                then emit(56)
                                else begin
                                       if( x.typ <> notyp ) and (y.typ <>
notyp)
                                       then error(32);
                                       x.typ := notyp
                                     end
                              end
                         else begin { op in [idiv,imod] }
                                if (x.typ = ints) and (y.typ = ints)
                                then if op = idiv
                                     then emit(58)
                                     else emit(59)
                                else begin
                                       if ( x.typ <> notyp ) and (y.typ <>
notyp)
                                       then error(34);
                                       x.typ := notyp
                                     end
                              end
              end { while }
          end { term };
        begin { simpleexpression }
          if sy in [plus,minus]
          then begin
                 op := sy;
                 insymbol;
                 term( fsys+[plus,minus],x);
```

```
                if x.typ > reals
                then error(33)
                else if op = minus
                     then emit(36)
             end
        else term(fsys+[plus,minus,orsy],x);
        while sy in [plus,minus,orsy] do
          begin
            op := sy;
            insymbol;
            term(fsys+[plus,minus,orsy],y);
            if op = orsy
            then begin
                   if ( x.typ = bools )and(y.typ = bools)
                   then emit(51)
                   else begin
                          if( x.typ <> notyp) and (y.typ <> notyp)
                          then error(32);
                          x.typ := notyp
                        end
                 end
            else begin
                   x.typ := resulttype(x.typ,y.typ);
                   case x.typ of
                     notyp: ;
                     ints: if op = plus
                           then emit(52)
                           else emit(53);
                     reals:if op = plus
                           then emit(54)
                           else emit(55)
                   end { case }
                 end
          end { while }
      end { simpleexpression };
begin { expression  }
  simpleexpression(fsys+[eql,neq,lss,leq,gtr,geq],x);
  if sy in [ eql,neq,lss,leq,gtr,geq]
  then begin
         op := sy;
         insymbol;
         simpleexpression(fsys,y);
         if(x.typ in [notyp,ints,bools,chars]) and (x.typ = y.typ)
         then case op of
                eql: emit(45);
                neq: emit(46);
                lss: emit(47);
                leq: emit(48);
                gtr: emit(49);
                geq: emit(50);
              end
         else begin
                if x.typ = ints
                then begin
                       x.typ := reals;
                       emit1(26,1)
                     end
                else if y.typ = ints
```

```
                            then begin
                                   y.typ := reals;
                                   emit1(26,0)
                                 end;
                      if ( x.typ = reals)and(y.typ=reals)
                      then case op of
                             eql: emit(39);
                             neq: emit(40);
                             lss: emit(41);
                             leq: emit(42);
                             gtr: emit(43);
                             geq: emit(44);
                           end
                      else error(35)
                    end;
                x.typ := bools
              end
      end { expression };

    procedure assignment( lv, ad: integer );
      var x,y: item;
          f  : integer;
      begin   { tab[i].obj in [variable,prozedure] }
x.typ := tab[i].typ;
x.ref := tab[i].ref;
        if tab[i].normal
        then f := 0
        else f := 1;
        emit2(f,lv,ad);
        if sy in [lbrack,lparent,period]
        then selector([becomes,eql]+fsys,x);
        if sy = becomes
        then insymbol
        else begin
               error(51);
               if sy = eql
               then insymbol
             end;
        expression(fsys,y);
        if x.typ = y.typ
        then if x.typ in stantyps
             then emit(38)
             else if x.ref <> y.ref
                  then error(46)
                  else if x.typ = arrays
                       then emit1(23,atab[x.ref].size)
                       else emit1(23,btab[x.ref].vsize)
        else if(x.typ = reals )and (y.typ = ints)
        then begin
               emit1(26,0);
               emit(38)
             end
        else if ( x.typ <> notyp ) and ( y.typ <> notyp )
             then error(46)
      end { assignment };

    procedure compoundstatement;
      begin
```

```
        insymbol;
        statement([semicolon,endsy]+fsys);
        while sy in [semicolon]+statbegsys do
          begin
            if sy = semicolon
            then insymbol
            else error(14);
            statement([semicolon,endsy]+fsys)
          end;
        if sy = endsy
        then insymbol
        else error(57)
      end { compoundstatement };

    procedure ifstatement;
var x : item;
          lc1,lc2: integer;
begin
        insymbol;
        expression( fsys+[thensy,dosy],x);
        if not ( x.typ in [bools,notyp])
        then error(17);
        lc1 := lc;
        emit(11);  { jmpc }
        if sy = thensy
        then insymbol
        else begin
               error(52);
               if sy = dosy
               then insymbol
             end;
        statement( fsys+[elsesy]);
        if sy = elsesy
        then begin
               insymbol;
               lc2 := lc;
emit(10);
               code[lc1].y := lc;
               statement(fsys);
               code[lc2].y := lc
             end
        else code[lc1].y := lc
end { ifstatement };

    procedure casestatement;
      var x : item;
i,j,k,lc1 : integer;
casetab : array[1..csmax]of
                     packed record
                       val,lc : index
                     end;
        exittab : array[1..csmax] of integer;

procedure caselabel;
        var lab : conrec;
k : integer;
        begin
          constant( fsys+[comma,colon],lab );
```

```
          if lab.tp <> x.typ
          then error(47)
          else if i = csmax
               then fatal(6)
               else begin
                      i := i+1;
k := 0;
                      casetab[i].val := lab.i;
                      casetab[i].lc := lc;
repeat
                        k := k+1
                      until casetab[k].val = lab.i;
                      if k < i
                      then error(1); { multiple definition }
                    end
        end { caselabel };

      procedure onecase;
        begin
          if sy in constbegsys
          then begin
                 caselabel;
                 while sy = comma do
                   begin
                     insymbol;
                     caselabel
                   end;
                 if sy = colon
                 then insymbol
                 else error(5);
                 statement([semicolon,endsy]+fsys);
j := j+1;
                 exittab[j] := lc;
emit(10)
               end
          end { onecase };
      begin  { casestatement  }
        insymbol;
        i := 0;
        j := 0;
        expression( fsys + [ofsy,comma,colon],x );
        if not( x.typ in [ints,bools,chars,notyp ])
then error(23);
        lc1 := lc;
        emit(12); {jmpx}
if sy = ofsy
        then insymbol
        else error(8);
        onecase;
        while sy = semicolon do
          begin
            insymbol;
            onecase
          end;
        code[lc1].y := lc;
        for k := 1 to i do
          begin
            emit1( 13,casetab[k].val);
```

```
            emit1( 13,casetab[k].lc);
          end;
        emit1(10,0);
        for k := 1 to j do
code[exittab[k]].y := lc;
if sy = endsy
        then insymbol
        else error(57)
      end { casestatement };

    procedure repeatstatement;
      var x : item;
          lc1: integer;
      begin
        lc1 := lc;
        insymbol;
        statement( [semicolon,untilsy]+fsys);
        while sy in [semicolon]+statbegsys do
          begin
            if sy = semicolon
            then insymbol
            else error(14);
            statement([semicolon,untilsy]+fsys)
          end;
        if sy = untilsy
        then begin
               insymbol;
               expression(fsys,x);
               if not(x.typ in [bools,notyp] )
               then error(17);
               emit1(11,lc1);
             end
        else error(53)
      end { repeatstatement };

    procedure whilestatement;
      var x : item;
          lc1,lc2 : integer;
      begin
        insymbol;
        lc1 := lc;
        expression( fsys+[dosy],x);
        if not( x.typ in [bools, notyp] )
then error(17);
        lc2 := lc;
        emit(11);
if sy = dosy
        then insymbol
        else error(54);
statement(fsys);
        emit1(10,lc1);
        code[lc2].y := lc
end { whilestatement };

    procedure forstatement;
      var  cvt : types;
x :  item;
          i,f,lc1,lc2 : integer;
```

```
begin
        insymbol;
        if sy = ident
        then begin
               i := loc(id);
               insymbol;
               if i = 0
               then cvt := ints
               else if tab[i].obj = vvariable
                    then begin
                           cvt := tab[i].typ;
                           if not tab[i].normal
                           then error(37)
else emit2(0,tab[i].lev, tab[i].adr );
if not ( cvt in [notyp, ints, bools, chars])
                           then error(18)
                         end
                    else begin
                           error(37);
                           cvt := ints
                         end
             end
        else skip([becomes,tosy,downtosy,dosy]+fsys,2);
        if sy = becomes
        then begin
               insymbol;
               expression( [tosy, downtosy,dosy]+fsys,x);
               if x.typ <> cvt
               then error(19);
             end
        else skip([tosy, downtosy,dosy]+fsys,51);
        f := 14;
        if sy in [tosy,downtosy]
        then begin
               if sy = downtosy
               then f := 16;
               insymbol;
               expression([dosy]+fsys,x);
               if x.typ <> cvt
               then error(19)
             end
        else skip([dosy]+fsys,55);
        lc1 := lc;
        emit(f);
        if sy = dosy
        then insymbol
        else error(54);
        lc2 := lc;
statement(fsys);
        emit1(f+1,lc2);
        code[lc1].y := lc
end { forstatement };

    procedure standproc( n: integer );
      var i,f : integer;
x,y : item;
      begin
        case n of
```

```
          1,2 : begin { read }
                  if not iflag
                  then begin
                         error(20);
                         iflag := true
                       end;
                  if sy = lparent
                  then begin
                         repeat
                           insymbol;
                           if sy <> ident
                           then error(2)
                           else begin
                                  i := loc(id);
                                  insymbol;
                                  if i <> 0
                                  then if tab[i].obj <> vvariable
                                       then error(37)
                                       else begin
                                              x.typ := tab[i].typ;
                                              x.ref := tab[i].ref;
                                              if tab[i].normal
then f := 0
                                              else f := 1;
                                              emit2(f,tab[i].lev,tab[i].adr);
if sy in [lbrack,lparent,period]
                                              then selector( fsys+
[comma,rparent],x);
                                              if x.typ in
[ints,reals,chars,notyp]
then emit1(27,ord(x.typ))
else error(41)
                                            end
                                end;
                           test([comma,rparent],fsys,6);
                         until sy <> comma;
                         if sy = rparent
                         then insymbol
                         else error(4)
                       end;
                  if n = 2
                  then emit(62)
                end;
          3,4 : begin { write }
                  if sy = lparent
                  then begin
                         repeat
                           insymbol;
                           if sy = stringcon
                           then begin
                                  emit1(24,sleng);
                                  emit1(28,inum);
                                  insymbol
                                end
                           else begin
expression(fsys+[comma,colon,rparent],x);
if not( x.typ in stantyps )
                                  then error(41);
```

```
                                  if sy = colon
                                  then begin
insymbol;
                                         expression( fsys+
[comma,colon,rparent],y);
if y.typ <> ints
                                         then error(43);
                                         if sy = colon
                                         then begin
                                                if x.typ <> reals
                                                then error(42);
                                                insymbol;
                                                expression(fsys+
[comma,rparent],y);
                                                if y.typ <> ints
                                                then error(43);
                                                emit(37)
                                              end
                                         else emit1(30,ord(x.typ))
                                       end
else emit1(29,ord(x.typ))
end
                        until sy <> comma;
                        if sy = rparent
                        then insymbol
                        else error(4)
                      end;
                if n = 4
                then emit(63)
              end; { write }
        end { case };
      end { standproc } ;
    begin { statement }
      if sy in statbegsys+[ident]
      then case sy of
             ident : begin
                       i := loc(id);
                       insymbol;
                       if i <> 0
                       then case tab[i].obj of
konstant,typel : error(45);
                                vvariable:       assignment(
tab[i].lev,tab[i].adr);
prozedure:       if tab[i].lev <> 0
                                                 then call(fsys,i)
                                                 else standproc(tab[i].adr);
                                funktion:        if tab[i].ref = display[level]
                                                 then assignment(tab[i].lev+1,0)
                                                 else error(45)
                              end { case }
                     end;
             beginsy : compoundstatement;
             ifsy    : ifstatement;
             casesy  : casestatement;
             whilesy : whilestatement;
             repeatsy: repeatstatement;
             forsy   : forstatement;
           end;  { case }
```

```
      test( fsys, [],14);
    end { statement };
  begin  { block }
    dx := 5;
    prt := t;
    if level > lmax
    then fatal(5);
    test([lparent,colon,semicolon],fsys,14);
    enterblock;
    prb := b;
    display[level] := b;
    tab[prt].typ := notyp;
    tab[prt].ref := prb;
    if ( sy = lparent ) and ( level > 1 )
    then parameterlist;
    btab[prb].lastpar := t;
    btab[prb].psize := dx;
    if isfun
    then if sy = colon
         then begin
                insymbol; { function type }
                if sy = ident
                then begin
                       x := loc(id);
                       insymbol;
                       if x <> 0
                       then if tab[x].typ in stantyps
                            then tab[prt].typ := tab[x].typ
                            else error(15)
                     end
                else skip( [semicolon]+fsys,2 )
              end
         else error(5);
    if sy = semicolon
    then insymbol
    else error(14);
    repeat
      if sy = constsy
      then constdec;
      if sy = typesy
      then typedeclaration;
      if sy = varsy
      then variabledeclaration;
      btab[prb].vsize := dx;
      while sy in [procsy,funcsy] do
        procdeclaration;
      test([beginsy],blockbegsys+statbegsys,56)
    until sy in statbegsys;
    tab[prt].adr := lc;
    insymbol;
    statement([semicolon,endsy]+fsys);
    while sy in [semicolon]+statbegsys do
      begin
        if sy = semicolon
        then insymbol
        else error(14);
        statement([semicolon,endsy]+fsys);
      end;
```

```
    if sy = endsy
    then insymbol
    else error(57);
    test( fsys+[period],[],6 )
  end { block };



procedure interpret;
  var ir : order ;         { instruction buffer }
      pc : integer;        { program counter }
      t  : integer;        { top stack index }
b  : integer;        { base index }
      h1,h2,h3: integer;
      lncnt,ocnt,blkcnt,chrcnt: integer;     { counters }
      ps : ( run,fin,caschk,divchk,inxchk,stkchk,linchk,lngchk,redchk );
fld: array [1..4] of integer;  { default field widths }
      display : array[0..lmax] of integer;
      s  : array[1..stacksize] of   { blockmark:     }
           record
             case cn : types of        { s[b+0] = fct result }
               ints : (i: integer );   { s[b+1] = return adr }
               reals :(r: real );      { s[b+2] = static link }
               bools :(b: boolean );   { s[b+3] = dynamic link }
               chars :(c: char )       { s[b+4] = table index }
end;

  procedure dump;
    var p,h3 : integer;
begin
      h3 := tab[h2].lev;
      writeln(psout);
      writeln(psout);
      writeln(psout,'       calling ', tab[h2].name );
      writeln(psout,'         level ',h3:4);
      writeln(psout,' start of code ',pc:4);
      writeln(psout);
      writeln(psout);
      writeln(psout,' contents of display ');
      writeln(psout);
      for p := h3 downto 0 do
        writeln(psout,p:4,display[p]:6);
      writeln(psout);
      writeln(psout);
      writeln(psout,' top of stack  ',t:4,' frame base ':14,b:4);
      writeln(psout);
      writeln(psout);
      writeln(psout,' stack contents ':20);
      writeln(psout);
      for p := t downto 1 do
        writeln( psout, p:14, s[p].i:8);
      writeln(psout,'< = = = >':22)
    end; {dump }

  procedure inter0;
    begin
      case ir.f of
        0 : begin { load addrss }
```

```
          t := t + 1;
          if t > stacksize
          then ps := stkchk
          else s[t].i := display[ir.x]+ir.y
        end;
    1 : begin  { load value }
          t := t + 1;
          if t > stacksize
          then ps := stkchk
          else s[t] := s[display[ir.x]+ir.y]
        end;
    2 : begin  { load indirect }
          t := t + 1;
          if t > stacksize
          then ps := stkchk
          else s[t] := s[s[display[ir.x]+ir.y].i]
        end;
    3 : begin  { update display }
          h1 := ir.y;
          h2 := ir.x;
          h3 := b;
          repeat
            display[h1] := h3;
            h1 := h1-1;
            h3 := s[h3+2].i
          until h1 = h2
        end;
    8 : case ir.y of
          0 : s[t].i := abs(s[t].i);
          1 : s[t].r := abs(s[t].r);
          2 : s[t].i := sqr(s[t].i);
          3 : s[t].r := sqr(s[t].r);
          4 : s[t].b := odd(s[t].i);
          5 : s[t].c := chr(s[t].i);
          6 : s[t].i := ord(s[t].c);
          7 : s[t].c := succ(s[t].c);
          8 : s[t].c := pred(s[t].c);
          9 : s[t].i := round(s[t].r);
          10 : s[t].i := trunc(s[t].r);
          11 : s[t].r := sin(s[t].r);
          12 : s[t].r := cos(s[t].r);
          13 : s[t].r := exp(s[t].r);
          14 : s[t].r := ln(s[t].r);
          15 : s[t].r := sqrt(s[t].r);
          16 : s[t].r := arcTan(s[t].r);
          17 : begin
                 t := t+1;
                 if t > stacksize
                 then ps := stkchk
                 else s[t].b := eof(prd)
               end;
          18 : begin
                 t := t+1;
                 if t > stacksize
                 then ps := stkchk
                 else s[t].b := eoln(prd)
               end;
        end;
```

```
        9 : s[t].i := s[t].i + ir.y; { offset }
      end { case ir.y }
    end; { inter0 }

procedure inter1;
    var h3, h4: integer;
begin
      case ir.f of
        10 : pc := ir.y ; { jump }
        11 : begin  { conditional jump }
               if not s[t].b
then pc := ir.y;
               t := t - 1
end;
        12 : begin { switch }
               h1 := s[t].i;
               t := t-1;
               h2 := ir.y;
               h3 := 0;
               repeat
                 if code[h2].f <> 13
                 then begin
                        h3 := 1;
                        ps := caschk
                      end
                 else if code[h2].y = h1
                      then begin
                             h3 := 1;
                             pc := code[h2+1].y
                           end
                      else h2 := h2 + 2
               until h3 <> 0
             end;
        14 : begin { for1up }
               h1 := s[t-1].i;
               if h1 <= s[t].i
               then s[s[t-2].i].i := h1
               else begin
                      t := t - 3;
                      pc := ir.y
                    end
             end;
        15 : begin { for2up }
               h2 := s[t-2].i;
               h1 := s[h2].i+1;
               if h1 <= s[t].i
               then begin
                      s[h2].i := h1;
                      pc := ir.y
                    end
               else t := t-3;
             end;
        16 : begin  { for1down }
               h1 := s[t-1].i;
               if h1 >= s[t].i
               then s[s[t-2].i].i := h1
               else begin
                      pc := ir.y;
```

```
                    t := t - 3
                  end
            end;
        17 : begin  { for2down }
               h2 := s[t-2].i;
               h1 := s[h2].i-1;
               if h1 >= s[t].i
               then begin
                      s[h2].i := h1;
                      pc := ir.y
                    end
               else t := t-3;
            end;
        18 : begin  { mark stack }
               h1 := btab[tab[ir.y].ref].vsize;
               if t+h1 > stacksize
               then ps := stkchk
               else begin
                      t := t+5;
                      s[t-1].i := h1-1;
                      s[t].i := ir.y
                    end
            end;
        19 : begin  { call }
               h1 := t-ir.y;  { h1 points to base }
               h2 := s[h1+4].i;  { h2 points to tab }
               h3 := tab[h2].lev;
               display[h3+1] := h1;
               h4 := s[h1+3].i+h1;
               s[h1+1].i := pc;
               s[h1+2].i := display[h3];
               s[h1+3].i := b;
               for h3 := t+1 to h4 do
                 s[h3].i := 0;
               b := h1;
               t := h4;
               pc := tab[h2].adr;
               if stackdump
               then dump
            end;
      end { case }
    end; { inter1 }

  procedure inter2;
    begin
      case ir.f of
        20 : begin   { index1 }
               h1 := ir.y;  { h1 points to atab }
               h2 := atab[h1].low;
               h3 := s[t].i;
               if h3 < h2
               then ps := inxchk
               else if h3 > atab[h1].high
                    then ps := inxchk
                    else begin
                           t := t-1;
                           s[t].i := s[t].i+(h3-h2)
                         end
```

```
          end;
       21 : begin  { index }
            h1 := ir.y ; { h1 points to atab }
            h2 := atab[h1].low;
            h3 := s[t].i;
            if h3 < h2
            then ps := inxchk
            else if h3 > atab[h1].high
                 then ps := inxchk
                 else begin
                        t := t-1;
                        s[t].i := s[t].i + (h3-h2)*atab[h1].elsize
                      end
          end;
       22 : begin  { load block }
            h1 := s[t].i;
            t := t-1;
            h2 := ir.y+t;
            if h2 > stacksize
            then ps := stkchk
            else while t < h2 do
                   begin
                     t := t+1;
                     s[t] := s[h1];
                     h1 := h1+1
                   end
          end;
       23 : begin  { copy block }
            h1 := s[t-1].i;
            h2 := s[t].i;
            h3 := h1+ir.y;
            while h1 < h3 do
              begin
                s[h1] := s[h2];
                h1 := h1+1;
                h2 := h2+1
              end;
            t := t-2
          end;
       24 : begin  { literal }
            t := t+1;
            if t > stacksize
            then ps := stkchk
            else s[t].i := ir.y
          end;
       25 : begin  { load real }
            t := t+1;
            if t > stacksize
            then ps := stkchk
            else s[t].r := rconst[ir.y]
          end;
       26 : begin  { float }
            h1 := t-ir.y;
            s[h1].r := s[h1].i
          end;
       27 : begin  { read }
            if eof(prd)
            then ps := redchk
```

```
               else case ir.y of
                      1 : read(prd, s[s[t].i].i);
                      2 : read(prd, s[s[t].i].r);
                      4 : read(prd, s[s[t].i].c);
                    end;
               t := t-1
             end;
        28 : begin   { write string }
h1 := s[t].i;
               h2 := ir.y;
               t := t-1;
               chrcnt := chrcnt+h1;
if chrcnt > lineleng
               then ps := lngchk;
               repeat
                 write(prr,stab[h2]);
                 h1 := h1-1;
                 h2 := h2+1
               until h1 = 0
             end;
        29 : begin  { write1 }
               chrcnt := chrcnt + fld[ir.y];
               if chrcnt > lineleng
               then ps := lngchk
               else case ir.y of
                      1 : write(prr,s[t].i:fld[1]);
                      2 : write(prr,s[t].r:fld[2]);
                      3 : if s[t].b
                          then write('true')
                          else write('false');
                      4 : write(prr,chr(s[t].i));
                    end;
               t := t-1
             end;
      end { case }
    end; { inter2 }

  procedure inter3;
    begin
      case ir.f of
        30 : begin { write2 }
               chrcnt := chrcnt+s[t].i;
               if chrcnt > lineleng
               then ps := lngchk
               else case ir.y of
                      1 : write(prr,s[t-1].i:s[t].i);
                      2 : write(prr,s[t-1].r:s[t].i);
                      3 : if s[t-1].b
                          then write('true')
                          else write('false');
                    end;
               t := t-2
             end;
        31 : ps := fin;
        32 : begin  { exit procedure }
               t := b-1;
               pc := s[b+1].i;
               b := s[b+3].i
```

```
            end;
       33 : begin  { exit function }
              t := b;
              pc := s[b+1].i;
              b := s[b+3].i
            end;
       34 : s[t] := s[s[t].i];
       35 : s[t].b := not s[t].b;
       36 : s[t].i := -s[t].i;
       37 : begin
              chrcnt := chrcnt + s[t-1].i;
              if chrcnt > lineleng
              then ps := lngchk
              else write(prr,s[t-2].r:s[t-1].i:s[t].i);
              t := t-3
            end;
       38 : begin  { store }
              s[s[t-1].i] := s[t];
              t := t-2
            end;
       39 : begin
              t := t-1;
              s[t].b := s[t].r=s[t+1].r
            end;
     end { case }
   end; { inter3 }

 procedure inter4;
   begin
     case ir.f of
       40 : begin
              t := t-1;
              s[t].b := s[t].r <> s[t+1].r
            end;
       41 : begin
              t := t-1;
              s[t].b := s[t].r < s[t+1].r
            end;
       42 : begin
              t := t-1;
              s[t].b := s[t].r <= s[t+1].r
            end;
       43 : begin
              t := t-1;
              s[t].b := s[t].r > s[t+1].r
            end;
       44 : begin
              t := t-1;
              s[t].b := s[t].r >= s[t+1].r
            end;
       45 : begin
              t := t-1;
              s[t].b := s[t].i = s[t+1].i
            end;
       46 : begin
              t := t-1;
              s[t].b := s[t].i <> s[t+1].i
            end;
```

```
        47 : begin
               t := t-1;
               s[t].b := s[t].i < s[t+1].i
             end;
        48 : begin
               t := t-1;
               s[t].b := s[t].i <= s[t+1].i
             end;
        49 : begin
               t := t-1;
               s[t].b := s[t].i > s[t+1].i
             end;
      end { case }
    end; { inter4 }

  procedure inter5;
    begin
      case ir.f of
        50 : begin
               t := t-1;
               s[t].b := s[t].i >= s[t+1].i
             end;
        51 : begin
               t := t-1;
               s[t].b := s[t].b or s[t+1].b
             end;
        52 : begin
               t := t-1;
               s[t].i := s[t].i+s[t+1].i
             end;
        53 : begin
               t := t-1;
               s[t].i := s[t].i-s[t+1].i
             end;
        54 : begin
               t := t-1;
               s[t].r := s[t].r+s[t+1].r;
             end;
        55 : begin
               t := t-1;
               s[t].r := s[t].r-s[t+1].r;
             end;
        56 : begin
               t := t-1;
               s[t].b := s[t].b and s[t+1].b
             end;
        57 : begin
               t := t-1;
               s[t].i := s[t].i*s[t+1].i
             end;
        58 : begin
               t := t-1;
               if s[t+1].i = 0
               then ps := divchk
               else s[t].i := s[t].i div s[t+1].i
             end;
        59 : begin
               t := t-1;
```

```
              if s[t+1].i = 0
              then ps := divchk
              else s[t].i := s[t].i mod s[t+1].i
            end;
      end { case }
    end; { inter5 }

  procedure inter6;
    begin
      case ir.f of
        60 : begin
               t := t-1;
               s[t].r := s[t].r*s[t+1].r;
             end;
        61 : begin
               t := t-1;
               s[t].r := s[t].r/s[t+1].r;
             end;
        62 : if eof(prd)
             then ps := redchk
             else readln;
        63 : begin
               writeln(prr);
               lncnt := lncnt+1;
               chrcnt := 0;
               if lncnt > linelimit
               then ps := linchk
             end
      end { case };
    end; { inter6 }
  begin { interpret }
    s[1].i := 0;
    s[2].i := 0;
    s[3].i := -1;
    s[4].i := btab[1].last;
    display[0] := 0;
    display[1] := 0;
    t := btab[2].vsize-1;
    b := 0;
    pc := tab[s[4].i].adr;
    lncnt := 0;
    ocnt := 0;
    chrcnt := 0;
    ps := run;
    fld[1] := 10;
    fld[2] := 22;
    fld[3] := 10;
    fld[4] := 1;
    repeat
      ir := code[pc];
      pc := pc+1;
      ocnt := ocnt+1;
      case ir.f div 10 of
0 : inter0;
        1 : inter1;
        2 : inter2;
        3 : inter3;
        4 : inter4;
```

```
5 : inter5;
        6 : inter6;
      end; { case }
    until ps <> run;

    if ps <> fin
    then begin
           writeln(prr);
           write(prr, ' halt at', pc :5, ' because of ');
           case ps of
             caschk  : writeln(prr,'undefined case');
             divchk  : writeln(prr,'division by 0');
             inxchk  : writeln(prr,'invalid index');
             stkchk  : writeln(prr,'storage overflow');
             linchk  : writeln(prr,'too much output');
             lngchk  : writeln(prr,'line too long');
             redchk  : writeln(prr,'reading past end or file');
           end;
           h1 := b;
           blkcnt := 10;    { post mortem dump }
           repeat
             writeln( prr );
             blkcnt := blkcnt-1;
             if blkcnt = 0
             then h1 := 0;
             h2 := s[h1+4].i;
             if h1 <> 0
             then writeln( prr, '',tab[h2].name, 'called at', s[h1+1].i:5);
             h2 := btab[tab[h2].ref].last;
             while h2 <> 0 do
               with tab[h2] do
                 begin
                   if obj = vvariable
                   then if typ in stantyps
                        then begin
                               write(prr,'',name,'=');
                               if normal
                               then h3 := h1+adr
                               else h3 := s[h1+adr].i;
                               case typ of
                                 ints : writeln(prr,s[h3].i);
                                 reals: writeln(prr,s[h3].r);
                                 bools: if s[h3].b
                                        then writeln(prr,'true')
                                        else writeln(prr,'false');
                                 chars: writeln(prr,chr(s[h3].i mod 64 ))
                               end
                             end;
                   h2 := link
                 end;
             h1 := s[h1+3].i
           until h1 < 0
         end;
    writeln(prr);
    writeln(prr,ocnt,' steps');
  end; { interpret }
```

```
procedure setup;
  begin
    key[1] := 'and       ';
    key[2] := 'array     ';
    key[3] := 'begin     ';
    key[4] := 'case      ';
    key[5] := 'const     ';
    key[6] := 'div       ';
    key[7] := 'do        ';
    key[8] := 'downto    ';
    key[9] := 'else      ';
    key[10] := 'end       ';
    key[11] := 'for       ';
    key[12] := 'function  ';
    key[13] := 'if        ';
    key[14] := 'mod       ';
    key[15] := 'not       ';
    key[16] := 'of        ';
    key[17] := 'or        ';
    key[18] := 'procedure ';
    key[19] := 'program   ';
    key[20] := 'record    ';
    key[21] := 'repeat    ';
    key[22] := 'then      ';
    key[23] := 'to        ';
    key[24] := 'type      ';
    key[25] := 'until     ';
    key[26] := 'var       ';
    key[27] := 'while     ';

    ksy[1] := andsy;
    ksy[2] := arraysy;
    ksy[3] := beginsy;
    ksy[4] := casesy;
    ksy[5] := constsy;
    ksy[6] := idiv;
    ksy[7] := dosy;
    ksy[8] := downtosy;
    ksy[9] := elsesy;
    ksy[10] := endsy;
    ksy[11] := forsy;
    ksy[12] := funcsy;
    ksy[13] := ifsy;
    ksy[14] := imod;
    ksy[15] := notsy;
    ksy[16] := ofsy;
    ksy[17] := orsy;
    ksy[18] := procsy;
    ksy[19] := programsy;
    ksy[20] := recordsy;
    ksy[21] := repeatsy;
    ksy[22] := thensy;
    ksy[23] := tosy;
    ksy[24] := typesy;
    ksy[25] := untilsy;
    ksy[26] := varsy;
    ksy[27] := whilesy;
```

```
    sps['+'] := plus;
    sps['-'] := minus;
    sps['*'] := times;
    sps['/'] := rdiv;
    sps['('] := lparent;
    sps[')'] := rparent;
    sps['='] := eql;
    sps[','] := comma;
    sps['['] := lbrack;
    sps[']'] := rbrack;
    sps['''] := neq;
    sps['!'] := andsy;
    sps[';'] := semicolon;
  end { setup };

procedure enterids;
  begin
    enter('          ',vvariable,notyp,0); { sentinel }
    enter('false     ',konstant,bools,0);
    enter('true      ',konstant,bools,1);
    enter('real      ',typel,reals,1);
    enter('char      ',typel,chars,1);
    enter('boolean   ',typel,bools,1);
    enter('integer   ',typel,ints,1);
    enter('abs       ',funktion,reals,0);
    enter('sqr       ',funktion,reals,2);
    enter('odd       ',funktion,bools,4);
    enter('chr       ',funktion,chars,5);
    enter('ord       ',funktion,ints,6);
    enter('succ      ',funktion,chars,7);
    enter('pred      ',funktion,chars,8);
    enter('round     ',funktion,ints,9);
    enter('trunc     ',funktion,ints,10);
    enter('sin       ',funktion,reals,11);
    enter('cos       ',funktion,reals,12);
    enter('exp       ',funktion,reals,13);
    enter('ln        ',funktion,reals,14);
    enter('sqrt      ',funktion,reals,15);
    enter('arctan    ',funktion,reals,16);
    enter('eof       ',funktion,bools,17);
    enter('eoln      ',funktion,bools,18);
    enter('read      ',prozedure,notyp,1);
    enter('readln    ',prozedure,notyp,2);
    enter('write     ',prozedure,notyp,3);
    enter('writeln   ',prozedure,notyp,4);
    enter('          ',prozedure,notyp,0);
  end;


begin  { main }
setup;
  constbegsys := [ plus, minus, intcon, realcon, charcon, ident ];
  typebegsys := [ ident, arraysy, recordsy ];
  blockbegsys := [ constsy, typesy, varsy, procsy, funcsy, beginsy ];
  facbegsys := [ intcon, realcon, charcon, ident, lparent, notsy ];
  statbegsys := [ beginsy, ifsy, whilesy, repeatsy, forsy, casesy ];
```

```
stantyps := [ notyp, ints, reals, bools, chars ];
lc := 0;
ll := 0;
cc := 0;
ch := ' ';
errpos := 0;
errs := [];
writeln( 'NOTE input/output for users program is console : ' );
writeln;
write( 'Source input file ?');
readln( inf );
assign( psin, inf );
reset( psin );
write( 'Source listing file ?');
readln( outf );
assign( psout, outf );
rewrite( psout );
assign ( prd, 'con' );
write( 'result file : ' );
readln( fprr );
assign( prr, fprr );
reset ( prd );
rewrite( prr );

t := -1;
a := 0;
b := 1;
sx := 0;
c2 := 0;
display[0] := 1;
iflag := false;
oflag := false;
skipflag := false;
prtables := false;
stackdump := false;

insymbol;

if sy <> programsy
then error(3)
else begin
       insymbol;
       if sy <> ident
       then error(2)
       else begin
              progname := id;
              insymbol;
              if sy <> lparent
              then error(9)
              else repeat
                     insymbol;
                     if sy <> ident
                     then error(2)
                     else begin
                            if id = 'input     '
                            then iflag := true
                            else if id = 'output    '
                                 then oflag := true
```

```
                                    else error(0);
                                insymbol
                            end
                        until sy <> comma;
                    if sy = rparent
                    then insymbol
                    else error(4);
                    if not oflag then error(20)
                end
        end;
  enterids;
  with btab[1] do
    begin
      last := t;
      lastpar := 1;
      psize := 0;
      vsize := 0;
    end;
  block( blockbegsys + statbegsys, false, 1 );
  if sy <> period
  then error(2);
  emit(31);  { halt }
  if prtables
  then printtables;
  if errs = []
  then interpret
  else begin
         writeln( psout );
         writeln( psout, 'compiled with errors' );
         writeln( psout );
         errormsg;
       end;
  writeln( psout );
  close( psout );
  close( prr )
end.
```