SEQFIL

```
<NIC-NLS>SEQFIL.NLS;18, 11-JAN-72 22:46 BLP ; ["count"];
FILE seqfil % L10 to <Nic-nls>Seqfil %

%.....declarations.....%

   REGISTER r1 = 1, r2 = 2, r3 = 3, r4 = 4, p = 7, wa = 8;
   REF prseqwk;
   DECLARE
      maxlev = 63, nlOfst = 40B, tenfst = 0,
      sfhfnm = 48, sfhpgn = 70,
      seqer = 4B8, seqeof = 1B9, maxtrc = 63, maxmclv = 1,
      ascmbk = 175B;

%.....output sequential, quickprint, and assembler.....%

   (outseq) PROCEDURE   % does the Output Sequential File command %
      (ofilnam,    % string containing name of the output file %
      da,          % display area descriptor %
      forceup);    % whether alphas are to be forced upper case %
      REF ofilnam, da;
      dismes(1, $"Output Sequential in Progress");
      opseqf (&ofilnam, &da, forceup, FALSE, opsqfl);
      dismes(0);
      RETURN;
      END.

   (outqp)  PROCEDURE   % does the Output Quickprint command %
      (ofilnam,    % string containing name of the output file %
      da);         % display area descriptor %
      REF ofilnam, da;
      dismes(1, $"Output Quickprint in Progress");
      opseqf (&ofilnam, &da, FALSE, TRUE, opqpfl);
      dismes(0);
      RETURN;
      END.

   (outasm) PROCEDURE   % does the Output to Assembler File command %
      (ofilnam,    % string containing name of the output file %
      da);         % display area descriptor %
      REF ofilnam, da;
      dismes(1, $"Output for Assembler in Progress");
      opseqf (&ofilnam, &da, FALSE, FALSE, opmcfl);
      dismes(0);
      RETURN;
      END.

   (opseqf) PROCEDURE
      (ofilnam,    % string containing name of the output file %
      da,          % display area descriptor %
      upflg,       % whether to force alphabetic characters uppercase %
      pgflg,       % whether to paginate %
      typfil);     % type of file %

      %This is the main control file for generating a sequential file.
      Basically, it uses the sequence generator to get new statements
      and then adds these statements to a sequential file.  The file is
```

```
both opened and closed here. %
%------------------%
LOCAL jfn, sw, toplev, dirnam;
LOCAL STRING string[200];
REF ofilnam, da, sw;

% open the output file %
   dirnam ← IF typfil=opqpfl THEN $prtdir ELSE 0;
   IF NOT jfn ←
      lgetjfn (dirnam, &ofilnam, $txtext, gtjoosf, $lit) THEN
         SIGNAL (ofilerr, $lit);
   IF NOT sysopen (jfn, write, chrtyp, $lit) THEN
      BEGIN
      reljfn (jfn);
      SIGNAL (ofilerr, $lit);
      END;
%initialize formatting stuff%
   sfucf  ← upflg;
   sfpgno ← pgflg;
   sflnel ← da.damcol/da.dahinc;
   sfndlv ← da.daind/da.dahinc;
   sfmxnd ← da.damind/da.dahinc;
%initialize sequence generator%
   &sw ← openseq (da.dacsp, endfil, da.davspec, da.davspc2,
   da.dausqcod, da.dacacode);
%initialize stuff for doing pmap's%
   sfpmr1.lhword ← 4B5;
   sfpmr1.rhword ← $sfbuff/1000B;
   sfpmr2.lhword ← jfn;
   sfpmr2.rhword ← 0;
   sfpmr3 ← 140001B6;
   sfbyte ← 0;
   sfbufl ← slngth(
      (sfbptr ← stbptr(empty) + $sfbuff),
      (sfndbf ← stbptr(5) + $sfbufe) );
IF sfnxtst(&sw) THEN
   BEGIN
   IF sfpgno THEN
      BEGIN
      bldsfhdr(&da);
      *string* ← EOL, *sfhead*, '1, 22B;
      sflnpg ← 3;
      sftrln(stbptr(empty)+$string+1,
      stbptr(string.L)+$string+1);
      BUMP sfpgno;
      END;
   IF typfil=opmcfl THEN
      BEGIN
      IF (toplev ← sw.swclvl) = 0 THEN toplev ← 1;
      sfmacpt(&da, &sw, toplev);
      END
   ELSE sfputst(&da, &sw, typfil);
   WHILE sfnxtst(&sw) DO
      IF typfil=opmcfl THEN sfmacpt(&da, &sw, toplev)
      ELSE sfputst(&da, &sw, typfil);
   END;
```

```
   % close sequence %
      closeseq (&sw);
   %close file%
      %pmap last page out%
         r1 ← sfpmr1;
         r2 ← sfpmr2;
         r3 ← sfpmr3;
         !JSYS pmap;
      %dismiss page from fork%
         r1 ← -1;
         r2 ← sfpmr1;
         r3 ← sfpmr3;
         !JSYS pmap;
      sfbyte ← sfbyte + slngth((stbptr(empty) + $sfbuff), sfbptr);
      r1.lhword ← 12B;
      r1.rhword ← jfn;
      r2 ← -1;
      r3 ← sfbyte;
      !JSYS chfdb;
      % close output file (but keep jfn) %
         IF NOT sysclose (jfn .V 4B11, $lit) THEN
            BEGIN
            reljfn (jfn);
            dismes (2, $lit);
            GOTO STATE;
            END;
      % delete old versions, if not Output Quickprint %
         IF typfil # opqpfl THEN delovsrns (jfn, nvtk);
      % releas the jfn %
         reljfn (jfn);
   RETURN;
   END.

(sfnxtst) PROCEDURE (sw);
   %This procedure calls the sequence generator to find the next
   statement.  If there are no more statements in the sequence, it
   returns FALSE.  Otherwise, it initializes CCPOS for reading the
   next statement, which may involve skipping the statement name. %
   %------------------%
   LOCAL TEXT POINTER tptr;
   REF sw;
   IF (tptr ← seqgen(&sw)) = endfil THEN
      RETURN(FALSE);
   tptr[1] ← IF sw.swvspec.vsnamf THEN 1
      ELSE fchtxt(getsdb(tptr));
   CCPOS tptr;
   RETURN(TRUE);
   END.

(sfputst) PROCEDURE (da, sw, typfil);
   %This copies the contents of READC to a local string. It expects
   READC to be set up to begin reading.  When it completes
   processing this statement it performs the necessary cleanup to
   begin processing the next statement (by calling SFENDST) and
   returns.%
   %------------------%
```

```
    LOCAL TEXT POINTER tptr;
    LOCAL curlin, chrcnt, indcnt, begbp, bytptr, char, tabpos, count,
    inblnk, outblnk, frstblnk, string[50];
    LOCAL STRING stnsig[30];
    REF da, sw;
    %initialize this statement%
       bytptr ← begbp ← stbptr(empty) + $string;
       indcnt ← IF NOT sw.swvspec.vsindf THEN 0
          ELSE MAX (0, MIN (sfndlv * (sw.swclvl-1), sfmxnd));
       IF (chrcnt ← indcnt) > 0 THEN
          IF typfil = opqpfl THEN
             BEGIN
             ↑bytptr ← 25B;   % multiple blanks %
             ↑bytptr ← indcnt;
             END
          ELSE FOR count ← 1 UP UNTIL > indcnt DO ↑bytptr ← SP;
       IF sw.swvspec.vsstnf AND NOT sw.swvspec.vsstnr
       AND sw.swcstid.stpsid # origin THEN
          BEGIN
          *stnsig* ← NULL;
          fechnm (sw.swsvw, $stnsig);
          FOR count ← empty + 1 UP UNTIL > stnsig.L DO
             ↑bytptr ← *stnsig*[count];
          ↑bytptr ← SP;
          chrcnt ← chrcnt + stnsig.L + 1;
          END;
       curlin ← 1;
    LOOP   % for each line %
       BEGIN
       inblnk ← CCPOS;
       outblnk ← frstblnk ← bytptr;
       LOOP
          CASE char ← READC OF
             = SP:
                BEGIN
                ↑bytptr ← SP;
                IF (chrcnt ← chrcnt + 1) >= sflnel THEN EXIT LOOP 1;
                outblnk ← bytptr;
                inblnk ← CCPOS;
                END;
             = ENDCHR: EXIT LOOP 2;
             = TAB:
                BEGIN
                tabpos ← MIN (sflnel,
                   fndtab (&da, chrcnt*da.dahinc) / da.dahinc);
                FOR chrcnt UP UNTIL >= tabpos DO
                   ↑bytptr ← SP;
                IF chrcnt >= sflnel THEN EXIT LOOP 1;
                outblnk ← bytptr;
                inblnk  ← CCPOS;
                END;
             = CR, =EOL:  EXIT LOOP 1;
             ENDCASE
                BEGIN
                ↑bytptr ← IF sfucf AND char IN ['a, 'z] THEN
                   char - 40B ELSE char;
```

```
                IF (chrcnt ← chrcnt + 1) >= sflnel THEN
                   BEGIN
                   IF frstblnk # outblnk THEN %there are blanks%
                      BEGIN
                      bytptr ← outblnk;
                      %reset readwk%
                         tptr ← sw.swstid;
                         tptr[1] ← inblnk;
                         CCPOS tptr;
                      END;
                   EXIT LOOP 1;
                   END;
                END;
          ↑bytptr ← IF char # CR THEN EOL ELSE CR;
          IF curlin >= sw.swvspec.vstrnc THEN EXIT LOOP 1;
          sftrln (begbp, bytptr);   % transfer line to output buffer %
          % intialize for next line %
             BUMP curlin;
             bytptr ← begbp;
             IF (chrcnt ← indcnt) > 0 THEN
                IF typfil = opqpfl THEN
                   BEGIN
                   ↑bytptr ← 25B;   % multiple blanks %
                   ↑bytptr ← indcnt;
                   END
                ELSE FOR count ← 1 UP UNTIL > indcnt DO ↑bytptr ← SP;
          END;
       % end of statement stuff %
       IF sw.swvspec.vsstnf AND sw.swvspec.vsstnr
       AND sw.swcstid.stpsid # origin THEN
          BEGIN  %statement number on right%
          *stnsig* ← NULL;
          fechnm (sw.swsvw, $stnsig);
          IF chrcnt + 1 + stnsig.L > sflnel THEN
             BEGIN %must go to new line%
             ↑bytptr ← EOL;
             sftrln (begbp, bytptr);   % transfer line to output buffer
             %
             bytptr ← begbp;
             chrcnt ← 0;
             END;
          IF typfil = opqpfl THEN
             BEGIN
             ↑bytptr ← 25B;   % multiple blanks %
             ↑bytptr ← MAX (0, sflnel - stnsig.L - chrcnt);
             END
          ELSE FOR count ← chrcnt + 1 UP UNTIL > sflnel - stnsig.L DO
             ↑bytptr ← SP;
          FOR count ← empty + 1 UP UNTIL > stnsig.L DO
             ↑bytptr ← *stnsig*[count];
          END;
       ↑bytptr ← EOL;
       sftrln (begbp, bytptr);   % transfer line to output buffer %
ELSE   IF sw.swvspec.vsblkf THEN
          BEGIN
          bytptr ← begbp;
          ↑bytptr ← EOL
          sftrln
          END
```

```
      IF sw.swvspec.vsidtf THEN
         BEGIN
         *stnsig* ← NULL;
         fechsig (sw.swcstid, $stnsig);
         IF typfil = opqpfl THEN
            BEGIN
            ↑bytptr ← 25B;    % multiple blanks %
            ↑bytptr ← MAX (0, sflnel - stnsig.L);
            END
         ELSE FOR count ← 1 UP UNTIL > sflnel - stnsig.L DO
            ↑bytptr ← SP;
         FOR count ← empty + 1 UP UNTIL > stnsig.L DO
            ↑bytptr ← *stnsig*[count];
         END;
      ↑bytptr ← EOL;
      sftrln (begbp, bytptr);    % transfer line to output buffer %
      END;
   RETURN;
   END.

(sfmacpt) PROCEDURE (da, sw, toplev);
   %This copies the contents of READC to a local string, for output
   to the assembler. It works like sfputst, except it inserts tabs
   at the beginning of statements that are of a lower level than the
   first statement in the series.%
   %------------------%
   LOCAL char, chrcnt, bytptr, count, begbp, outblnk, frstblnk,
   inblnk, string[50];
   LOCAL TEXT POINTER tptr;
   REF da, sw;
   LOOP
      BEGIN
      begbp ← bytptr ← stbptr(empty) + $string;
      CASE (count ← sw.swclvl - toplev) OF
         <0: count ← 0;
         >2: count ← 2; %two levels of indentation for macro file%
         ENDCASE;
      IF (chrcnt ← count) > 0 THEN
         DO ↑bytptr ← TAB UNTIL (count ← count - 1) <= 0;
      inblnk ← CCPOS;
      outblnk ← frstblnk ← bytptr;
      WHILE chrcnt <= sflnel DO
         CASE char ← READC OF
            = SP:
               BEGIN
               ↑bytptr ← char;
               BUMP chrcnt;
               outblnk ← bytptr;
               inblnk ← CCPOS;
               END;
            = ENDCHR: EXIT LOOP 1;
            = CR, =EOL:
               BEGIN
               ↑bytptr ← CR;
               IF char = EOL THEN ↑bytptr ← LF;
               inblnk ← CCPOS;
```

```
                outblnk ← bytptr;
                EXIT LOOP 1;
                END;
             ENDCASE
                BEGIN
                ↑bytptr ← char;
                BUMP chrcnt;
                END;
       IF char = ENDCHR THEN %null line%
          BEGIN
          IF .begbp = SP AND slngth(begbp, bytptr) = 1 THEN
             bytptr ← begbp;
          END
       ELSE
          IF frstblnk # outblnk %blanks in line%
          AND bytptr # outblnk THEN %last character not blank%
             BEGIN
             bytptr ← outblnk;
             %reset readwk%
                tptr ← sw.swstid;
                tptr[1] ← inblnk;
                CCPOS tptr;
             END;
       CASE char OF
          =CR, =ENDCHR, =EOL: NULL;
          ENDCASE
             BEGIN
             ↑bytptr ← CR;
             ↑bytptr ← LF;
             END;
       IF char = ENDCHR THEN EXIT LOOP 1;
       sftrln (begbp, bytptr);   % transfer line to output buffer %
       END;
    ↑bytptr ← CR;
    ↑bytptr ← LF;
    sftrln (begbp, bytptr);   % transfer line to output buffer %
    RETURN;
    END.

(sftrln) PROCEDURE (begbp, bytptr);
    %This routine copies characters bounded by begbp and bytptr (both
    byte pointers) into sfbuff and pmaps sfbuff into the print file
    when the buffer is full.%
    %------------------%
    LOCAL STRING string[100];
    UNTIL begbp = bytptr DO
       BEGIN
       ↑sfbptr ← ↑begbp;
       IF sfbptr = sfndbf THEN
          BEGIN
          %pmap page out%
             r1 ← sfpmr1;
             r2 ← sfpmr2;
             r3 ← sfpmr3;
             !JSYS pmap;
          %dismiss page from fork%
```

```
            r1 ← -1;
            r2 ← sfpmr1;
            r3 ← sfpmr3;
            !JSYS pmap;
         sfbptr ← stbptr(empty) + $sfbuff;
         BUMP sfpmr2;
         sfbyte ← sfbyte + sfbufl;
         END;
      END;
   IF sfpgno AND (sflnpg ← sflnpg + 1) >= 60 THEN
      BEGIN
      *string* ← 14B, EOL, *sfhead*, STRING(sfpgno), 22B;
      sflnpg ← 3;
      sftrln (stbptr(empty)+$string+1, stbptr(string.L) +
      $string+1);
      BUMP sfpgno;
      END;
   RETURN;
   END.

(bldsfhdr) PROCEDURE (da);
   %This routine builds a page header for output quickprint.%
   %------------------%
   LOCAL vspwrd, b;
   REF da;
   vspwrd ← da.davspec;
   %put initials into sfhead%
      *sfhead* ← "   ", *initsr*, ",   ";
   getdat($sfhead);
   %put truncation and level values into sfhead%
      IF vspwrd.vstrnc < maxtrc OR vspwrd.vslev < maxlev THEN
         BEGIN
         IF vspwrd.vstrnc < maxtrc THEN
            *sfhead* ← *sfhead*, " T=", STRING(vspwrd.vstrnc), ',
         ELSE *sfhead* ← *sfhead*, " T=ALL,";
         IF vspwrd.vslev < maxlev THEN
            *sfhead* ← *sfhead*, " L=", STRING(vspwrd.vslev), ',
         ELSE *sfhead* ← *sfhead*, " L=ALL,";
         END;
   %blank fill to position for file name%
      *sfhead* ← *sfhead*, 25B, MAX (0, b ← sfhfnm - sfhead.L);
   %get file name%
      filnam(da.dacsp.stfile, $sfhead);
   %blank fill to position for page number%
      *sfhead* ← *sfhead*, 25B, MAX (0, sfhpgn - (sfhead.L - 2) -
      b);
   RETURN;
   END.

%.....insert sequential.....%

   (inseq) PROCEDURE   % does the Input Sequential command %
      (stid,     % statement after which to insert %
      levstg,    % level relative to stid %
      ifilnam,   % string containing the name of the input file %
      filtyp);   % type of input %
```

```
% This is the main control routine for insert sequential.  The
input file is opened and closed here. %
%------------------%
LOCAL count, levblk, lstblk, curblk, curdpth, levchg, char, pos,
offset, jfn;
LOCAL STRING inbuff[150];
REF levstg, ifilnam;

dismes(1, $"Insert Sequential in Progress");
% open the input file %
   IF NOT jfn ← lgetjfn (0, &ifilnam, $txtext, gtjoisf, $lit)
   THEN
      SIGNAL (ofilerr, $lit);
   IF NOT sysopen (jfn, read, chrtyp, $lit) THEN
      BEGIN
      reljfn (jfn);
      SIGNAL (ofilerr, $lit);
      END;
levblk ← lstblk ← curdpth ← 0;
offset ← IF filtyp = n4Ofil THEN n4Ofst ELSE tenfst;
*lit* ← NULL;
isread (jfn, $inbuff, CR);
pos ← empty + 1;
WHILE *inbuff*[pos] = LF DO BUMP pos;
CASE *inbuff*[pos]  OF
   =ascmbk:
      levblk ← lstblk ← *[inbuff]*[pos ← pos + 1] - offset;
   =SP, =TAB:
      BEGIN
      levblk ← 1;
      LOOP
         CASE *inbuff*[pos ← pos + 1] OF
            =SP, =TAB: BUMP levblk;
            ENDCASE EXIT LOOP;
      lstblk ← levblk;
      pos ← pos - 1;
      END;
   ENDCASE pos ← pos - 1;
curdpth ← 0;
LOOP
   BEGIN
   UNTIL (pos := pos + 1) > inbuff.L DO
   CASE char ← *inbuff*[pos] OF
      =ascmbk:
         BEGIN
         count ← *inbuff*[pos ← pos + 1] - offset;
         UNTIL (count := count - 1) = 0 DO
         *lit* ← *lit*, SP;
         END;
      = LF: NULL;
      = CR: %end of statement%
         BEGIN
         stid ← <STRMNP, instat> (stid, $lit, &levstg);
         *lit* ← NULL;
         IF NOT isread(jfn, $inbuff, CR) THEN EXIT LOOP 2;
```

```
pos ← empty + 1;
WHILE *inbuff*[pos] = LF DO BUMP pos;
CASE *inbuff*[pos] OF
   %determine level of next statement%
   =ascmbk:
      curblk ← *inbuff*[pos ← pos + 1] - offset;
   =SP, =TAB:
      BEGIN
      count ← 1;
      LOOP
         CASE *inbuff*[pos ← pos + 1] OF
            =SP, =TAB:
               BEGIN
               IF filtyp = macfil AND count >= maxmclv
               THEN EXIT LOOP;
               BUMP count;
               END;
            ENDCASE EXIT LOOP;
      pos ← pos - 1;
      curblk ← count;
      END;
   ENDCASE
      BEGIN
      curblk ← 0;
      pos ← pos - 1;
      END;
IF curblk THEN
   BEGIN
   IF NOT levblk THEN levblk ← curblk;
   IF filtyp = macfil THEN
      BEGIN
      IF curdpth THEN *levstg* ← NULL
      ELSE
         BEGIN
         curdpth ← 1;
         *levstg* ← 'D;
         END;
      END
   ELSE
      CASE lstblk OF
         =curblk: %same level%
            *levstg* ← NULL;
         < curblk: %down a level%
            IF (curblk - lstblk) >= levblk
            OR curblk/levblk > curdpth THEN
               BEGIN
               *levstg* ← 'D;
               BUMP curdpth;
               END
            ELSE *levstg* ← NULL;
         ENDCASE %up a level%
            BEGIN
            levchg ← (lstblk - curblk) / levblk;
            IF levchg < 1 AND curdpth > 2 THEN
               levchg ← (curdpth - 1) - curblk/levblk;
            IF levchg > curdpth THEN levchg ← curdpth;
```

```
                        curdpth ← curdpth - levchg;
                        *levstg* ← NULL;
                        WHILE (levchg ← levchg - 1) >= 0 DO
                           *levstg* ← 'U;
                        END;
                  END
               ELSE
                  BEGIN
                  *levstg* ← NULL;
                  IF lstblk THEN
                     WHILE (curdpth := curdpth - 1) > 0 DO
                        *levstg* ← 'U;
                  curdpth ← 0;
                  END;
               lstblk ← curblk;
               END;
            ENDCASE *lit* ← *lit*, char;
         IF NOT isread(jfn, $inbuff, CR) THEN EXIT LOOP;
         pos ← empty;
         END;
      IF lit.L NOT= empty THEN
         <STRMNP, instat> (stid, $lit, &levstg);
      %close file%
         IF NOT sysclose (jfn, $lit) THEN
            BEGIN
            reljfn (jfn);
            dismes (2, $lit);
            GOTO STATE;
            END;
      dismes(0);
      RETURN;
      END.

(isread)
   %This procedure is used during insert sequential to read another
   character from the input file.%
   %------------------%
   PROCEDURE(jfn, inbuff, lstchr);
   LOCAL error, bytptr;
   REF inbuff;
   r2 ← bytptr ← chbptr(empty) + &inbuff;
   r3 ← inbuff.M;
   r4 ← lstchr;
   r1 ← jfn;
   !JSYS sin;
   inbuff.L ← slngth(bytptr, r2);
   r1 ← jfn;
   !JSYS gtsts;
   error ← r2;
   IF error .A seqer # 0 THEN
      BEGIN
      IF NOT sysclose (jfn, $lit) THEN
         BEGIN
         reljfn (jfn);
         err ($lit);
         END
```

count, maxcnt
count ← empty;
maxcnt ← inbuff.M
LOOP
BEGIN
r1 ← jfn
!JSYS bin
IF(r2 . A 377B) % Null byte % = 0
THEN EXIT LOOP;
↑bytptr ← r2;
IF(count ← count + 1) ≥ maxcnt THEN
BEGIN
inbuff.L ← count;
RETURN(TRUE);
END;
IF(r2 = lstchr OR (r2 = EOL
AND lstchr = CR)

```
      ELSE err ($"Input File Error");
      END;
   IF error .A seqeof # 0 THEN
      BEGIN
      IF inbuff.L > empty + 1 THEN RETURN(TRUE) ELSE RETURN(FALSE);
      END;
   RETURN(TRUE);
   END.

%.....output device and output processor.....%

(outdev) PROCEDURE    % does the Output Device command %
   (ofilnam,    % string containing the name of the output file %
   da,          % display area descriptor %
   device);     % whitch device %

   LOCAL jfn, dirnam, errors;
   LOCAL TEXT POINTER tp;
   REF ofilnam, da;

   % open the output file %
      dirnam ← IF device=opprdv THEN $prtdir ELSE 0;
      IF NOT jfn ←
         lgetjfn (dirnam, &ofilnam, $txtext, gtjoosf, $lit) THEN
            SIGNAL (ofilerr, $lit);
      IF NOT sysopen (jfn, write,
         CASE device OF
            = opfrdv: fr80typ;
            = optydv: lpttype;
            ENDCASE chrtyp,     $lit) THEN
         BEGIN
         reljfn (jfn);
         SIGNAL (ofilerr, $lit);
         END;
   opinit (&da, jfn, device);
   tp[0] ← da.dacsp; tp[1] ← 1;
   errors ← processor
      (IF exp THEN $xopname ELSE $opname, $tp, &da, odtyp, jfn);
   % close output file %
      IF NOT sysclose (jfn, $lit) THEN
         BEGIN
         reljfn (jfn);
         dismes (2, $lit);
         GOTO STATE;
         END;
   RETURN;
   END.

(outcmp) PROCEDURE    % does the Output Compiler command %
   (cmpnam,     % string containing the name of the compiler %
   ofilnam,     % string containing the name of the output file %
   da);         % display area descriptor %

   LOCAL jfn, errors;
   LOCAL TEXT POINTER tp;
   REF cmpnam, ofilnam, da;
```

```
    % open the output file %
       IF NOT jfn ← lgetjfn (0, &ofilnam, $relext, gtjoosf, $lit)
       THEN
          SIGNAL (ofilerr, $lit);
       IF NOT sysopen (jfn, write, bintyp, $lit) THEN
          BEGIN
          reljfn (jfn);
          SIGNAL (ofilerr, $lit);
          END;
    tp[0] ← da.dacsp; tp[1] ← 1;
    errors ← processor (&cmpnam, $tp, &da, cmptyp, jfn);
    % close output file (but keep jfn) %
       IF NOT sysclose (jfn .V 4B11, $lit) THEN
          BEGIN
          reljfn (jfn);
          dismes (2, $lit);
          GOTO STATE;
          END;
    % delete old versions, if no errors and no rubout %
       IF errors = 0 AND NOT inptrf THEN delovsrns (jfn, nvtk - 1);
       reljfn (jfn);
    RETURN (errors);
    END.

%.....processor dispatch.....%

    (processor) PROCEDURE   % start and run a processor %
       (prcnam,  % string containing the name of the processor %
       tp,       % text pointer to first thing to feed the processor %
       da,       % display area descriptor of the input to processor %
       type,     % type of the processor %
       outdsg);  % output designator:  either the address of a buffer or
       the jfn of an output file %

       LOCAL stid, prcjfn, size, error, count, savpreg, savwareg, vspec;
       LOCAL TEXT POINTER fchartp, tptr;
       LOCAL STRING ssysnam[40];
       REF prcnam, tp, da;

       % set up to get first character for the processor %
          vspec ← da.davspec;
          IF type = cmptyp THEN vspec.vsstnf ← TRUE;
             % turn statement numbers on for MPL %
          &prseqwk  ← openseq (tp, endfil, vspec, da.davspc2,
          da.dausqcod, da.dacacode);
          IF (stid ← seqgen(&prseqwk)) = endfil THEN
             BEGIN
             closeseq (&prseqwk);
             SIGNAL (prcerr, $"No processor input");
             END;
          IF tp.stastr OR type = upgtyp THEN FIND (tp ↑fchartp)
          ELSE FIND (SF(stid) ↑fchartp);
       IF NOT (prcjfn ← lgetjfn($subdir, &prcnam, $savext, gtjprf,
       $lit))
          THEN SIGNAL (prcerr, $lit);
```

```
% get a string containing name of the processor %
   IF type = upgtyp THEN *ssysnam* ← "NLSLLO"
   ELSE
      BEGIN
      jfntostr (prcjfn, $ssysnam, 11B9);
      IF NOT FIND SF(*ssysnam*) "SUBSYS" ↑tptr THEN
         *ssysnam* ← "PRVPRC"
      ELSE *ssysnam* ← tptr SE(*ssysnam*);
      END;
% set the subsystem name %
   r1 ← getsbn ($ssysnam);
   !JSYS setnm;
% "get" the processor %
   r1.lhword ← 4B5;
   r1.rhword ← prcjfn;
   !JSYS get;
IF [prcadr + 1] THEN chntab[9] ← [prcadr + 1] .V 1B6;
savpreg ← p; savwareg ← wa;  % save wa and p registers since the
compilers and the Output Processor will clobber them %
CASE type OF
   = upgtyp:   % compile a user program %
      BEGIN
      dismes (1, $"Compiling user program");
      swork ← fchartp; swork[1] ← fchartp[1]; fechcl(1, $swork);
      %swork[2] contains byte count+1, swork[4] has byte pointer%
      error ← [[prcadr]]
         (-1, $prgetst, outdsg, $psym, $levllc, swork[4],
         swork[2]-1 :size);
      END;
   = cmptyp:   % compile a program into an output file %
      BEGIN
      dismes (1, $"Compiler in progress");
      swork ← fchartp; swork[1] ← fchartp[1]; fechcl(1, $swork);
      %swork[2] contains byte count+1, swork[4] has byte pointer%
      error ← [[prcadr]]
         (outdsg, $prgetst, prseqwk.swsvw, , $levllc, swork[4],
         swork[2]-1);
      END;
   = odtyp:    % output device type %
      BEGIN
      dismes (1, $"Output Processor in progress");
      CCPOS fchartp;
      error ← [[prcadr]] ($oprwrk, $levllc);
      END;
   ENDCASE   % some other type of processor %
      BEGIN
      dismes (1, $"Processor in progress");
      swork ← fchartp; swork[1] ← fchartp[1]; fechcl(1, $swork);
      %swork[2] contains byte count+1, swork[4] has byte pointer%
      error ← [[prcadr]] (outdsg, $prgetst, , , $levllc,
      swork[4], swork[2]-1);
      END;
p ← savpreg; wa ← savwareg;  % restore wa and p registers %
entvec[0] ← 254B9 .V $init; %JRST init%
entvec[1] ← 254B9 .V $rentr; %JRST rentr%
r1 ← 4B5;
```

```
    r2 ← 2B6 .V $entvec;
    !JSYS sevec; %reset entry vector back to NLS%
    chntab[9] ← $stkovr .V 1B6;
    r1 ← nlssbn;
    !JSYS setnm;
    dismes (0);
    closeseq (&prseqwk);
    FOR count ← $bfree/1000B UP UNTIL >= $efree/1000B DO
       BEGIN
       r1 ← -1;
       r2.lhword ← 4B5;
       r2.rhword ← count;
       !JSYS pmap;
       END;
    IF inptrf THEN %rubout%
       BEGIN
       inptrf ← 0;
       clrbuf (1);
       IF type # upgtyp THEN
          IF NOT <IOEXEC, sysclose> (outdsg, $lit) THEN
             dismes (2, $lit);
       SIGNAL(-4, $"User Terminated Process");
       END;
    RETURN (error, size);
    END.

(prgetst) PROCEDURE;
    % returns byte pointer + count for next statement %
    LOCAL stid, sdb;
    REF sdb;
    IF (stid ← seqgen(&prseqwk)) = endfil THEN RETURN(endfil);
    lodsdb(getsdb(stid) : &sdb);
    RETURN(&sdb + sdbhdl + 4407B8, sdb.schars);
    END.

(oprchr) PROCEDURE;
     RETURN(READC);
     END.

(oprnst) PROCEDURE;
    LOCAL stid;
    IF (stid ← seqgen(&prseqwk)) # endfil THEN CCPOS SF(stid);
    RETURN (stid);
    END.

(oprtxt) PROCEDURE;
    LOCAL TEXT POINTER tp;
    tp ← prseqwk.swcstid;
    tp[1] ← fchtxt(getsdb(tp));
    CCPOS tp;
    RETURN;
    END.

(oprgps) PROCEDURE;
    RETURN (CCPOS);
    END.
```

IF prseqwk.swstid.stastv
THEN RETURN;

```
(oprrst) PROCEDURE (pos);
   LOCAL TEXT POINTER tp;
   tp ← swork;
   IF pos = -1 THEN FIND SE(tp) >
   ELSE
      BEGIN
      tp[1] ← pos;
      CCPOS tp;
      END;
   RETURN;
   END.

(oprlev) PROCEDURE;
   RETURN (getlev (prseqwk.swcstid));
   END.

(oprsvc) PROCEDURE;
   stvect (prseqwk.swcstid, prseqwk.swsvw);
   RETURN (prseqwk.swsvw);
   END.

(oprsig) PROCEDURE;
   fechsig (prseqwk.swcstid, $lit);
   RETURN ($lit);
   END.

(oprhdf) PROCEDURE;
   RETURN (getfhd (prseqwk.swcstid));
   END.

(opinit) PROCEDURE (da, jfn, devtyp);
   oprwrk ← jfn;
   oprwrk[1] ← devtyp;
   oprwrk[2] ← da;
   oprwrk[3] ← $oprchr;
   oprwrk[4] ← $oprtxt;
   oprwrk[5] ← $oprrst;
   oprwrk[6] ← $oprgps;
   oprwrk[7] ← $oprlev;
   oprwrk[8] ← $oprsvc;
   oprwrk[9] ← $oprsig;
   oprwrk[10] ← $oprhdf;
   oprwrk[11] ← $oprnst;
   oprwrk[12] ← IF nlmode = typewriter THEN FALSE ELSE TRUE;
   RETURN;
   END.

(getsbn) PROCEDURE (astrng);
   %convert string passed to sixbit name%
   LOCAL sixbit, charct, bytptr;
   REF astrng;
   bytptr ← 6B8 + $sixbit - 1;
   sixbit ← 0;
   charct ← empty;
   WHILE ((charct ← charct + 1) <= 6) AND (charct <= astrng.L) DO
```

```
    ↑bytptr ← (*astrng*/charct/ + 40B) .A 77B;
  RETURN(sixbit);
  END.

DECLARE psym=(
  2, "ps", ps, %start position for BETWEEN%
  2, "pe", pe, %end position for BETWEEN%
  6, "savpos", savpos, %BETWEEN prolog%
  6, "respos", respos, %BETWEEN epilog%
   6, "scndir", scndir, %scan direction for Find constructs%
   5, "swork", swork, %Readc string work area%
   6, "sworkl", sworkl,
   3, "pdc", pdc, %pointer decrement%
   4, "flag", flag, %flag set by Find statements%
   5, "tstsr", tstsr, %test string%
   3, "bfs", bfs, %branch false and scan -- for unanchored
  patterns%
   4, "tstf", tstf, %unanchored string search%
   5, "cpfse", cpfse, %statement end -- only for use by compiler%
  4, "cctf", cctf, %unanchored character class test%
  3, "cct", cct, %anchored character class test%
  4, "tchf", tchf, %unanchored character test%
   5, "sptr1", sptr1, %string construction pointer%
   5, "sptr2", sptr2, %string construction pointer%
   3, "bsc", bsc, %begin string construction routine%
   3, "esc", esc, %end string construction routine%
   5, "apchr", apchr, %append a character to a string routine%
   4, "apsr", apsr, %append a string to a string routine%
   6, "srlset", srlset, %set string to single character%
   5, "cpysr", cpysr, %copy string to string routine%
   6, "asrref", asrref, %create reference to string for use in text
  pointer%
   3, "kps", kps, %copy string to string construction area%
   6, "aptstr", aptstr, %append t-string (defined by two t-ptrs) to
  string routine%
   7, "modeset", modeset, %mode flag for string construction%
   7, "mkrstay", mkrstay, %marker flag for string construction%
  6, "sysrtn", sysrtn, %RETURN; code%
  5, "input", input, %NLS input character routine%
  5, "lookc", lookc, %look at next input char (without inputting
  it) routine%
  4, "echo", echo, %echo a string (in response to user input)
  routine%
  4, "crlf", crlf, %output a CR followed by a LF%
  6, "prompt", prompt, %type a prompt string to the user%
  6, "typeas", typeas, %type a string to the user%
  6, "dismes", dismes, %display a message for the user%
  5, "todco", todco, %type a char with translation - TNLS%
  6, "echoff", echoff, %turn monitor echoing off%
  5, "echon", echon, %turn monitor echoing on%
  4, "tbug", tbug, %TNLS bug selection routine%
  5, "rdlit", rdlit, %read a TNLS literal%
  6, "rnmdel", rnmdel, %right name delimiter field of SDB header%
  6, "inpcuc", inpcuc, %input a char from user and force upper
  case%
  6, "printg", printg, %TNLS print group routine%
```

```
3, "xrs", xrs, %executor routine for Replace statement%
3, "cdp", cdp, %Core-NLS Delete Plex routine%
3, "cis", cis, %Core-NLS Insert Statement routine%
6, "lookup", lookup, %routine used for content, word, name, SID
searches%
4, "open", open, %file open routine%
6, "openwk", openwk, %work-file open routine%
5, "close", close, %file close routine%
6, "updtfl", updtfl, %update file routine%
6, "getnxt", getnxt, %get stid of next statement routine%
6, "getsub", getsub, %get stid of first sub-statement routine%
6, "getsuc", getsuc, %get stid of successor statement routine%
6, "stfile", stfile, %file number field of an stid%
6, "stpsid", stpsid, %PSID number field of an stid%
6, "stastr", stastr, %whether a stid points to an a-string%
5, "stadr", stadr, %address of the a-string field in a stid %
4, "stwc", stwc,
5, "stblk", stblk,
6, "stpsdb", stpsdb,
5, "stsid", stsid,
6, "endfil", endfil, %END OF FILE stid%
6, "origin", origin, %PSID of origin statement%
5, "empty", empty, %length of a string with no characters in it%
5, "readc", readc, %READC routine%
5, "ldchr", ldchr, %get a char from a string routine%
6, "repchr", repchr, %replace a char in a string%
6, "sysmrt", sysmrt, %RETURN of multiple results%
6, "sysovr", sysovr, %stack overflow code%
5, "chrct", chrct, %character class test%
6, "fechcl", fechcl, %READC setup routine%
6, "sys2rt", sys2rt, %RETURN of 2 results%
6, "srmake", srmake, %STRING (exp) routine%
6, "chpair", chpair, %append substring%
4, "srmk", srmk, %STRING (exp) to string area%
9, "modeshift", modeshift, %mode flag for string construction%
5, "ascom", ascom, %string comparison routine ?%
5, "inbug", inbug, %BUG routine in INPUT constructs%
6, "instid", instid, %STID routine in INPUT constructions%
8, "inlevadj", inlevadj, %LEVADJ routine in INPUT constructs%
6, "intext", intext, %TEXT routine in INPUT constructs%
6, "inname", inname, %NAME routine in INPUT constructs%
5, "innum", innum, %NUM routine in INPUT constructs%
4, "insr", insr, %STRING routine in INPUT constructs%
5, "inword", inword, %WORD routine in INPUT constructs%
6, "invspc", invspc, %VIEWSPECS routine in INPUT constructs%
4, "inch", inch, %CHARACTER routine in INPUT constructs%
3, "err", err, %error routine%
6, "syssig", syssig, %system signal routine%
6, "sysgnl", sysgnl, %system signal number%
6, "sysmsg", sysmsg, %system signal message%
7, "ckident", ckident, %check IDENT routine%
7, "rdident", rdident, %Read IDENT routine%
9, "identlist", identlist, %Read IDENTLIST routine%
6, "scnlft", scnlft, %scan left%
6, "scnrht", scnrht, %scan right%
3, "pcp", pcp, %push character position%
```

```
4, "sptr", sptr, %store pointer%
3, "fcp", fcp, %fix character position%
7, "fxswork", fxswork, %fix swork%
3, "scp", scp, %set character position%
6, "begarb", begarb, %arbitrary number prolog%
6, "incarb", incarb, %arbitrary number increment%
6, "endarb", endarb, %arbitrary number epilog%
5, "fxsp1", fxsp1, %store text pointer into spfr1%
5, "fxsp2", fxsp2, %store text pointer into spfr2%
3, "lpr", lpr, %load pointer into ac's%
4, "dptr", dptr, %deposit pointer from ac's%
4, "span", span, %arbitrary number of character class%
5, "idtst", idtst, %test for ID=signature%
6, "txtdat", txtdat, %get date for statement%
6, "apachr", apachr, %append a character to LIT string%
6, "getids", getids, % IDENTFILE manipulation %
6, "getiid", getiid,
7, "getiadd", getiadd,
7, "getinam", getinam,
9, "getilname", getilname,
7, "getiaff", getiaff,
7, "getiexp", getiexp,
7, "getimem", getimem,
8, "geticord", geticord,
10, "getiverify", getiverify,
8, "getiuser", getiuser,
8, "getihost", getihost,
7, "getiimp", getiimp,
9, "getiphone", getiphone,
12, "getifunction", getifunction,
14, "geticapability", geticapability,
10, "getisubcol", getisubcol,
12, "getidelivery", getidelivery,
10, "getimcmnts", getimcmnts,
7, "expdtst", expdtst,
7, "jgrptst", jgrptst,
6, "afftst", afftst,
7, "afgptst", afgptst,
8, "idstatus", idstatus,
7, "jidsbot", jidsbot,
6, "jidstk", jidstk,
5, "rcpsh", rcpsh,
5, "rstsk", rstsk,
5, "rcpto", rcpto,
% Stuff Walt needed %
   1, "p", p,
   2, "r1", r1,
   2, "r2", r2,
   2, "wa", wa,
   3, "cea", cea,
   3, "get", get,
   4, "crng", crng,
   4, "csdb", csdb,
   4, "halt", halt,
   4, "init", init,
   4, "pmap", pmap,
```

```
    5, "bfree", bfree,
    5, "efree", efree,
    5, "rentr", rentr,
    5, "sevec", sevec,
    5, "write", write,
    6, "chntab", chntab,
    6, "chrtyp", chrtyp,
    6, "ckstrc", ckstrc,
    6, "entvec", entvec,
    6, "getdat", getdat,
    6, "gtjprf", gtjprf,
    6, "levllc", levllc,
    6, "lhword", lhword,
    6, "nfstid", nfstid,
    6, "opinit", opinit,
    6, "opname", opname,
    6, "opprdv", opprdv,
    6, "oprwrk", oprwrk,
    6, "prcadr", prcadr,
    6, "reljfn", reljfn,
    6, "rhword", rhword,
    6, "savext", savext,
    6, "stdvsp", stdvsp,
    6, "subdir", subdir,
    6, "txtext", txtext,
    7, "gtjoosf", gtjoosf,
    7, "lgetjfn", lgetjfn,
    7, "sysopen", sysopen,
    8, "namelook", namelook,
    8, "sysclose", sysclose,
% User Program Stuff %
    5, "gpllO", gpllO,
    7, "gpgmrst", gpgmrst,
    7, "gpconan", gpconan,
    6, "gpinst", gpinst,
    8, "gpdeinst", gpdeinst,
    7, "gpexpgm", gpexpgm,
    5, "gppop", gppop,
    8, "gpstatus", gpstatus,
    6, "upgcnv", upgcnv,
%   stuff for handling command feedback in user pgms%
    6, "nlmode", nlmode,
   11, "fulldisplay", fulldisplay,
    6, "inlsca", inlsca,
    6, "in2sca", in2sca,
    6, "recred", recred,
    6, "plxset", plxset,
    6, "getctc", getctc,
    3, "mlf", mlf,
    4, "mlfl", mlfl,
    6, "cfldsp", cfldsp,
% Sort-Merge-Update primitives%
    6, "cgsort", cgsort,
    7, "cgmerge", cgmerge,
    8, "cgupdate", cgupdate,
% Sequence generator routines %
```

```
    7, "openseq", openseq,   % open a sequence %
    8, "closeseq", closeseq,   % close a sequence %
    6, "seqgen", seqgen,   % get next item in a sequence %
    5, "sport", sport,   % "port-send return mechanism %
    4, "send", send,   % "port-send return mechanism %
    5, "cpysw", cpysw,  % copies right parts of one sw to another
    %
    5, "sqopn", sqopn,   % user seqgen call as openseq %
    5, "sqcls", sqcls,   % user seqgen call as closeseq %
    6, "sqgnxt", sqgnxt,   % user seqgen call as seqgen %
% Sequence generator work area record fields %
    6, "swstid", swstid, % stid of current item; may be endfil or
    a stastr %
    8, "swlbstid", swlbstid,  % STID of statement heading last
    branch in the sequence %
    7, "swcstid", swcstid,   % stid of current statement %
    7, "swvspec", swvspec,   % first word of viewspecs %
    6, "swvsp2", swvsp2,   % second word of viewspecs %
    5, "swsvw", swsvw,   % address of statement vector work area %
    6, "swclvl", swclvl, % level of current statement %
    8, "swusqcod", swusqcod,  % address of user sequence generator
    code or 0 %
    8, "swcacode", swcacode,  % address of content anylyzer code
    or 0 %
% display area descriptor record fields %
    3, "lda", lda,  %returns address of display area%
    7, "davspec", davspec,  %first viewspec word%
    7, "davspc2", davspc2,  %second viewspec word%
    5, "dacsp",  dacsp,     %csp for this da%
    6, "daccnt", daccnt,   %character count for dacsp t-pointer%
    6, "datab0", datab0,   %first tab position%
    6, "datab1", datab1,   %second tab position%
    6, "datab2", datab2,   %third tab position%
    6, "datab3", datab3,   %fourth tab position%
    6, "datab4", datab4,   %fifth tab position%
    6, "datab5", datab5,   %sixth tab position%
    6, "datab6", datab6,   %seventh tab position%
    6, "datab7", datab7,   %eighth tab position%
    6, "datab8", datab8,   %nineth tab position%
    6, "datab9", datab9,   %tenth tab position%
    8, "dacacode", dacacode,   % address of content analyzer
    program -- or 0 %
    8, "dausqcod", dausqcod,   % address of sequence generator
    program -- or 0 %
    9, "daukeycod", daukeycod, % address of sort key extractor
    program -- or 0 %
% viewspecs record fields %
    5, "vslev",  vslev,     %lower level bound--level clipping%
    6, "vstrnc", vstrnc,    %line truncation value%
    6, "vsrlev", vsrlev,    %relative level%
    6, "vslevd", vslevd,    %direction of relative level
    adjustment%
    6, "vscapf", vscapf,    %content analyzer pass-flag%
    6, "vscakf", vscakf,    %content analyzer k viewspec flag%
    6, "vsusqf", vsusqf,    %user sequence generator flag%
    6, "vsbrof", vsbrof,    %branch only flag%
```

```
        6, "vsplxf", vsplxf,    %plex-only flag%
        6, "vsblkf", vsblkf,    %blank line flag%
        6, "vsindf", vsindf,    %indenting on flag%
        6, "vsnamf", vsnamf,    %names on flag%
        6, "vsstnf", vsstnf,    %loc-nums on flag%
        6, "vsstnr", vsstnr,    %loc-nums on right flag%
        6, "vsaflf", vsaflf,    %abbreviated feedback line flag%
        6, "vsmkrf", vsmkrf,    %display markers flag%
        6, "vsfrzf", vsfrzf,    %frozen statements flag%
        6, "vsidtf", vsidtf,    %initials, date, and time flag%
        6, "vspagf", vspagf,    %page flag for tty output%
        6, "vsdaft", vsdaft);    %display area format flag%

   DECLARE psyme[20];
FINISH of SEQFIL
\Catalog) Catalog of NLS Procedures Called
   (auxcod,opseqf) (utilty,stbptr) (auxcod,sfnxtst) (auxcod,bldsfhdr)
   (auxcod,sfnewp) (auxcod,sfmacpt) (auxcod,sfputst) (seqgen,fechnm)
   (dspgen,fndtab) (auxcod,sftran) (auxcod,sfendst) (auxcod,getdat)
   (ioexec,filnam) (auxcod,isread) (filmnp,fchsdb) (filmnp,lodrng)
   (utilty,xtrnam) (txcmnd,astruc) (utilty,hash) (ioexec,getctx)
   (ioexec,setctx) (auxcod,nmdlsset) (auxcod,xnmdlset) (strmnp,plxset)
   (strmnp,grptst)
   (txcmnd,clrbuf) (dspgen,dismes) (inpfbk,input) (ioexec,delfil)
   (auxcod,resnls) (auxcod,thwrfp) (inpfbk,zap) (inpfbk,gabtgt)
   (inpfbk,typeas) (tsprt,typeno) (auxcod,halt) (auxcod,oprgps)
   (dspgen,getrta) (ioexec,closeall) (auxcod,closms) (filmnp,fchtxt)
   (filmnp,getsdb) (seqgen,fechsn) (seqgen,getlev) (seqgen,stvect)
   (seqgen,fechsig) (filmnp,getfhd) (utilty,chbptr) (utilty,slngth)
   (auxcod,getsbn) (coren1,crerr) (auxcod,werr) (filmnp,filhdr)
   (utilty,mvbfbf) (txcmnd,tshptr) (inpfbk,crlf) (auxcod,dtfrmt)
(fcat) Formatted Catalog Branch
   (dtfrmt)  (auxcod,dtfrmt)
   (crlf)  (inpfbk,crlf)
   (tshptr)  (txcmnd,tshptr)
   (mvbfbf)  (utilty,mvbfbf)
   (filhdr)  (filmnp,filhdr)
   (werr)  (auxcod,werr)
   (crerr)  (coren1,crerr)
   (getsbn)  (auxcod,getsbn)
   (slngth)  (utilty,slngth)
   (chbptr)  (utilty,chbptr)
   (getfhd)  (filmnp,getfhd)
   (fechsig)  (seqgen,fechsig)
   (stvect)  (seqgen,stvect)
   (getlev)  (seqgen,getlev)
   (fechsn)  (seqgen,fechsn)
   (getsdb)  (filmnp,getsdb)
   (fchtxt)  (filmnp,fchtxt)
   (closms)  (auxcod,closms)
   (closeall)  (ioexec,closeall)
   (getrta)  (dspgen,getrta)
   (oprgps)  (auxcod,oprgps)
   (halt)  (auxcod,halt)
   (typeno)  (tsprt,typeno)
   (typeas)  (inpfbk,typeas)
```

```
(gabtgt)  (inpfbk,gabtgt)
(zap)  (inpfbk,zap)
(thwrfp)  (auxcod,thwrfp)
(resnls)  (auxcod,resnls)
(delfil)  (ioexec,delfil)
(input)  (inpfbk,input)
(dismes)  (dspgen,dismes)
(clrbuf)  (txcmnd,clrbuf)
(grptst)  (strmnp,grptst)
(plxset)  (strmnp,plxset)
(xnmdlset)  (auxcod,xnmdlset)
(nmdlsset)  (auxcod,nmdlsset)
(setctx)  (ioexec,setctx)
(getctx)  (ioexec,getctx)
(hash)  (utilty,hash)
(astruc)  (txcmnd,astruc)
(xtrnam)  (utilty,xtrnam)
(lodrng)  (filmnp,lodrng)
(fchsdb)  (filmnp,fchsdb)
(isread)  (auxcod,isread)
(filnam)  (ioexec,filnam)
(getdat)  (auxcod,getdat)
(sfendst)  (auxcod,sfendst)
(sftran)  (auxcod,sftran)
(fndtab)  (dspgen,fndtab)
(fechnm)  (seqgen,fechnm)
(sfputst)  (auxcod,sfputst)
(sfmacpt)  (auxcod,sfmacpt)
(sfnewp)  (auxcod,sfnewp)
(bldsfhdr)  (auxcod,bldsfhdr)
(sfnxtst)  (auxcod,sfnxtst)
(stbptr)  (utilty,stbptr)
(opseqf)  (auxcod,opseqf)
```

SERGEN

```
<NLS>SEQGEN.NLS;49, 29-DEC-71 16:37 BLP ;
FILE seqgen  %  L10 to <rel-nls>seqgen  %

% Routines making up the sequence generator %

   % OPENSEQ is used to open a sequence
     CLOSESEQ is used to close a sequence
     SEQGEN is used to get the next statement in a sequence
     There is also a routine SEQNXT which gets the next statement in a
   sequence considering all the viewspecs except the content analyzer
   ones %

   SET sqsksz = 1000B;  % seqgen stack size %
   REGISTER  s=9, m=10;
   REF sqsavsw;   % global for saving argument (address of sequence work
   area) while switching stacks %

% the Sequence Generator %

   (openseq) PROCEDURE    % open a sequence %
      (fstid,   % STID with which to begin the sequence %
      lbstid,   % STID of statement heading last branch in the sequence%
      vspec,    % first word of viewspecs to use %
      vspc2,    % 2nd word of viewspecs %
      usqcod,   % address of user seqgen routine -- or 0 %
      cacode);  % address of CONAN routine  -- or 0 %

      % OPENSEQ
         Gets a sequence work area (with its attendent stack and
         statement vector work area) and returns the address of that
         work area.
         Initializes the work area
         If generating statement numbers, initializes the statement
         vector work area also.
         Perhaps calls the user's seqgen (with the address of the work
         area and telling it that its being called as openseq)
         The stack is not switched which means that user openseq is not
         allowed to do port-sends. %

      LOCAL sw;
      REF sw;

      % allocate a work area, a stack, and a statement vector work area%
         &sw ← getsgw();
      % initialize the work area %
         IF fstid.stastr  THEN   % stid points to a string %
            BEGIN
            sw.swstid ← sw.swcstid ← fstid;
            sw.swclvl ← 0;
            sw.swvspec ← vspec;
            sw.swvsp2  ← vspc2;
            END
         ELSE
            BEGIN
            IF getsid(fstid) = 0 THEN   % illegal stid %
               BEGIN
```

```
            sw.swcstid.stfile ← fstid.stfile;
            sw.swcstid.stpsid ← origin;
            sw.swstid ← endfil;
            END
         ELSE sw.swstid ← sw.swcstid ← fstid;
         % set up level and statement number stuff %
         sw.swclvl ←
            IF vspec.vsstnf THEN stvect (sw.swcstid, sw.swsvw)
            ELSE getlev (sw.swcstid);
         % set up viewspecs -- maybe relative level stuff %
            sw.swvspec ←
               IF vspec.vsrlev THEN reslev(vspec, sw.swclvl) ELSE
               vspec;
            sw.swvsp2 ← vspc2;
         END;
      sw.swlbstid ← lbstid;
      sw.swusqcod ← usqcod;
      sw.swcacode ← cacode;
      sw.swslvl ← sw.swclvl;   % remember what level started at %
      sw.swkflg ← FALSE;   % nothing has been sent or passed yet %
      % start seqgen coroutine first time at its initialization port,
      i.e., CALL it %
         sw.swcall ← TRUE;
   % maybe call the users seqgen code as openseq %
      IF vspec.vsusqf AND sw.swusqcod THEN
         [sw.swusqcod] (&sw, sqopn);
   RETURN (&sw);
   END.

(closeseq) PROCEDURE   % close a sequence %
   (sw);   % address of the sequence work area to close %

   % CLOSESEQ
      Check for a legal sequence work area.
      Perhaps calls the user's seqgen (with the address of the work
      area and telling it that its being called as a closeseq)
      Releases the sequence work area (with its attendent stack and
      statement vector work area)
      The stack is not switched which means the user closeseq is not
      allowed to do port-sends %
   REF sw;
   % maybe call the users seqgen code as closeseq %
      IF sw.swvspec.vsusqf AND sw.swusqcod THEN
         [sw.swusqcod] (&sw, sqcls);
   % release the work area, stack, and statement vector work area %
      relsgw (&sw);
   RETURN;
   END.

(seqgen) PROCEDURE
   (sw);   % address of a sequence generator work area %

   % SEQGEN
      Calling SEQGEN results in the caller being port-sent (looks
      like a normal return to the caller) the next "STID" in that
      sequence -- actually it may be an ENDFIL (no more items in this
```

```
      sequence), or a stastr, or the STID of a statement.
      Also the following fields of the work area are updated:
         SWCSTID (remains unchanged if an ENDFIL or stastr is the
         item returned)
         SWSTID (will be same as thing returned)
         SWSRSAV and SWMRSAV; SWCLVL; SWCALL, SWKFLG
         (unless conan code is using sport rather than using send or
         returning with flag set).
      It checks for a legal sequence work area.
      It also updates the statement vector if necessary.
      SEQGEN is actually merely a dispatch routine -- it calls either
      usqcod (user sequence code) or SSEQGEN (system sequence
      generator);
      It effects one-half of the coroutine linkage involved in
      port-sends (SPORT effects the other half).
      The input fork is informed that the Sequence Generator will
      handle rubouts.  Since the only way out of here is thru SPORT,
      the occurance of a rubout is checked for there and an ENDFIL
      sent if one occured.
      The stacks are switched here.  Thus no locals are allowed and
      the argument must be saved in a global. %

   REF sw;
   IF NOT (&sw IN [$sqgwas, $sqgaend]
      AND ((&sw - $sqgwas) MOD $sqwrkl) = 0 AND sw.swalloc) THEN
         SIGNAL (sqerr, $"illegal sequence work area");
   IF sw.swstid = endfil THEN RETURN (endfil);
   rubabt ← FALSE;  % tells input fork not to abort if see rubout %
   &sqsavsw ← &sw;       % save the argument in a global %
   % switch stacks %
      s ← sqsavsw.swsrsav := s;
      m ← sqsavsw.swmrsav := m;
   % depending on swcall (TRUE if last item in sequence was recieved
   by a normal return, FALSE it was recieved via a port-send), the
   next item in sequence is asked for by a CALL or a RETURN.  Note
   that OPENSEQ sets swcall TRUE and this is the only other place in
   NLS where it is tested or set. %
      IF sqsavsw.swcall := FALSE THEN
         BEGIN
         % call a user seqgen or SSEQGEN %
         IF sqsavsw.swusqcod AND sqsavsw.swvspec.vsusqf THEN
            [sqsavsw.swusqcod] (&sqsavsw, sqgnxt)
         ELSE sseqgen (&sqsavsw);
         sqsavsw.swcall ← TRUE;     % only get here if normal return
         made (not a port-send) %
         sport (&sqsavsw);     % SPORT will switch the stacks back and
         return to SEQGEN's caller %
         END
      ELSE RETURN;   % thru port, not to SEQGEN's caller %;
   END.

(sseqgen) PROCEDURE    % NOBODY BUT SEQGEN SHOULD CALL THIS ROUTINE %
   (sw);   % address of a sequence generator work area %

   % SSEQGEN
      Given address of sequence work area this procedure returns the
```

```
      next item in that sequence.
      SSEQGEN or SEQNXT or SEND or ucacod stores that value in SWSTID
      (and SWCSTID if it is not ENDFIL or a stastr).
      The routine SEQNXT finds the next STID, considering all
      viewspecs except content anlysis ones -- SSEQGEN takes care of
      those.
      Note that SEQNXT is not called the first time thru this
      routine.  Thus the first stid returned will be the one the
      sequence was initialized with -- unless conan code fails it.
      Rubouts are checked for after every call on conan code. %

   LOCAL stid;
   REF sw;
   LOOP
      BEGIN
      WHILE (sw.swvspec.vscapf OR (sw.swvspec.vscakf AND NOT
      sw.swkflg)) AND sw.swcacode > 0 AND sw.swstid # endfil DO
         % (conan must pass statements viewspec is on OR (turn off
         conan after first passed statement viewspec is on BUT
         haven't passed one yet)) AND there is a conan program AND we
         haven't reached the end of the sequence yet %
         BEGIN
         stid ← sw.swcstid;
         FIND SF(stid) ↑bl;
         [sw.swcacode] (&sw);
         IF flag OR inptrf %rubout% THEN
            BEGIN
            sw.swstid ← stid;  % a send from this same statement will
            have blown this field %
            sport (&sw);
            END;
         seqnxt (&sw);
         END;
      sport (&sw);
      seqnxt (&sw);
      END;
   END.

(seqnxt) PROCEDURE
   (sw);   % address of sequence work area %

   % SEQNXT
      Returns the next STID in this sequence (or ENDFIL if no more).
      It updates the following fields of the work area:
         swstid, swcstid (if not ENDFIL), and swclvl.
      It also updates the statement vector if necessary.
      It takes into account all viewspecs, except content analysis,
      during the search. %

   LOCAL stid, %current stid%
      vspec;   %first viewspec word (sw.swvspec)%
   REF sw;
   IF sw.swcstid.stastr THEN RETURN (sw.swstid ← endfil);
   vspec ← sw.swvspec;
   IF sw.swclvl >= vspec.vslev
      OR (stid ← getsub (sw.swcstid)) = sw.swcstid THEN
```

```
      BEGIN %see if superstructure fits viewspecs%
      IF (stid ← sw.swcstid) = sw.swlbstid
         OR vspec.vsbrof AND sw.swclvl <= sw.swslvl THEN
            RETURN (sw.swstid ← endfil);
      WHILE getftl (stid) DO
         BEGIN
         stid ← getsuc (stid);
         BUMP DOWN sw.swclvl;
         IF stid = sw.swlbstid
            OR vspec.vsbrof AND sw.swclvl <= sw.swslvl
            OR vspec.vsplxf AND sw.swclvl < sw.swslvl THEN
               RETURN (sw.swstid ← endfil);
         IF vspec.vsstnf THEN stvmod (sw.swsvw, up);
         END;
      IF stid.stpsid = origin THEN RETURN (sw.swstid ← endfil);
      %successor is next PSID in sequence%
      stid ← getsuc (stid);
      IF vspec.vsstnf THEN stvmod (sw.swsvw, suc);
      END
   ELSE
      BEGIN   %substructure fits viewspecs%
      BUMP sw.swclvl;   %increase level%
      IF vspec.vsstnf THEN stvmod (sw.swsvw, sub);
      END;
   RETURN (sw.swcstid ← sw.swstid ← stid);
   END.

(send) PROCEDURE
   (sw,    % address of a sequence work area %
   str);   % an ENDFIL or address of a string %

   % SEND
      Takes the place of the old send.
      Sets the swstid (but not the swcstid) field of the work area.
      Calls sport. %

   REF sw, str;
   IF &str = endfil THEN sw.swstid ← endfil
   ELSE
      BEGIN
      sw.swstid.stastr ← TRUE;
      sw.swstid.stadr  ← &str;
      IF str.L = empty THEN *str* ← SP; %no null strings permitted%
      END;
   sport (&sw);
   RETURN;
   END.

(sport)  PROCEDURE    % send-port mechanism %
   (sw);    % address of a sequence work area %

   % SPORT
      Effects one-half of the coroutine linkage involved in
      port-sends (SEQGEN effects the other half).
      Control o's and rubouts are checked for here.
      The stacks are switched here. %
```

```
   REF sw;
   IF NOT (&sw IN (*$sqgwas, $sqgaend)
      AND ((&sw - $sqgwas) MOD $sqwrkl) = 0 AND sw.swalloc) THEN
         SIGNAL (sqerr, $"illegal sequence work area");
   sw.swkflg ← TRUE; % something has been returned in this sequence %
   % the stacks are about to be switched so save everything needed
   later in globals %
      &sqsavsw ← &sw;
   % switch stacks %
      s ← sqsavsw.swsrsav := s;
      m ← sqsavsw.swmrsav := m;
   IF inptrf THEN   % a contol o or a rubout has occurred %
      BEGIN
      rubabt ← TRUE;
      sqsavsw.swstid ← endfil;
      END;
   % the following RETURN has the effect of returning from the
   procedure last called from the now-in-effect stack %
   RETURN (sqsavsw.swstid);
   END.

(sqinit) PROCEDURE;   % sequence generator init %

   % SQINIT
      Is only called at INIT time.
      It initializes all the sequence work areas (including the
      attached stacks and sequence vector work areas). %

   LOCAL sw, swcnt;    % address of a sequence generator work area %
   REF sw;
   swcnt ← 0;
   FOR &sw ← $sqgwas UP $sqwrkl UNTIL >= $sqgaend DO
      BEGIN
      sw.swalloc ← FALSE;   % allocation bit %
      sw.swsvw ← $sqsvws + (swcnt * (svmxlev + 1));
      (*$sqstks + (swcnt * $sqsksz)) ← $uflow;
      BUMP swcnt;
      END;
   RETURN;
   END.

(getsgw) PROCEDURE;   % get a sequence generator work area %

   % GETSGW
      Allocates, initializes some parts of, and returns the address
      of a sequence generator work area (a sequence work area, a
      statement vector work area, and a stack). %

   LOCAL sw, swcnt;    % address of a sequence generator work area %
   REF sw;
   swcnt ← 0;
   FOR &sw ← $sqgwas UP $sqwrkl UNTIL >= $sqgaend DO
      IF NOT sw.swalloc THEN
         BEGIN
         sw.swalloc ← TRUE;   % allocation bit %
```

```
            sw.swsrsav.rh ← sw.swmrsav.rh ← $sqstks + (swcnt * $sqsksz);
            sw.swsrsav.lh ← sw.swmrsav.lh ← -$sqsksz;
            RETURN (&sw);
            END
         ELSE BUMP swcnt;
      SIGNAL (sqerr, $"no more sequence work areas");
      END.

   (relsgw) PROCEDURE    % release a sequence generator work area %
      (sw);      % address of a sequence generator work area %

      % RELSGW
         Deallocates a sequence generator work area (a sequence work
         area, a statement vector work area, and a stack). %

      REF sw;
      IF NOT (&sw IN ($sqgwas, $sqgaend)
         AND ((&sw - $sqgwas) MOD $sqwrkl) = 0 AND sw.swalloc) THEN
            SIGNAL (sqerr, $"illegal sequence work area");
      sw.swalloc ← FALSE;
      RETURN;
      END.

% SEQGEN, STATEMENT NUMBER / VECTOR / SIGNATURE UTILITY ROUTINES %

   (cpysw) PROCEDURE  % copy parts of one sequence work area to another%
      (fromsw, tosw);   % record pointers to work areas %
         % this has the effect of making the tosw point to the same item
         as the fromsw %
      REF fromsw, tosw;

      tosw.swcstid ← fromsw.swcstid;
      tosw.swstid  ← fromsw.swstid;
      tosw.swvspec ← fromsw.swvspec;
      tosw.swvsp2  ← fromsw.swvsp2;
      tosw.swsvw   ← fromsw.swsvw;
      tosw.swclvl  ← fromsw.swclvl;
      RETURN;
      END.

   (reslev) PROCEDURE(vspec, clevel);
      %Given a viewspec word and a level, this routine will perform any
      relative level adjustment required by the viewspec word and return
      the updated viewspec word.%
      IF vspec.vsrlev THEN
         BEGIN
         IF vspec.vslevd THEN % up %
            vspec.vslev ← MAX (clevel-vspec.vslev, 1)
         ELSE % down %
            vspec.vslev ← MIN (clevel+vspec.vslev, 63);
         vspec.vsrlev ← FALSE;
         vspec.vslevd ← FALSE;
         END;
      RETURN(vspec);
      END.
```

```
(stvect)
   %This routine generates a statement vector.
   It is called with a STID as the first argument and the
   address of the first word of a 17 word work area as the
   second.
      The number of words in the work area determine how many
      levels down in structure the routine can go (currently
      17).  ERR(5) is called if the work area is not long
      enough to contain the vector for the given STID.
      STVECT uses routine STPOS.
   STVECT returns an integer, which is the level of the
   statement.
   It also creates a statement vector.  Upon completion, the
   first word of the work area contains the level of the STID
   and subsequent cells contain the position within the
   respective plexes.  Thus for statement 1D2, the vector would
   contain (3, 1, 4, 2) in the first four cells.%
   %-----------------%
   PROCEDURE (stid, stvwrk);
   LOCAL
      curwrd, %current word being built in vector%
      level;  %level of STID%
   REF stvwrk, curwrd;
   &curwrd ← &stvwrk + svmxlev;
   level ← 0;
   UNTIL stid.stpsid = origin DO %count up to origin%
      BEGIN
      IF (&curwrd ← &curwrd-1) <= &stvwrk THEN err(5);
      curwrd ← stpos(stid);
      stid ← getup(stid);
      BUMP level;
      END;
   stvwrk ← level;
   %move vector to top of work area%
   mvbfbf (&curwrd, &stvwrk+1, level);
   RETURN (level);
   END.


(stvmod)
   %This routine modifies a statement vector.  It is called
   with the address of a statement vector as the first
   argument, and a type parameter as the second.
   The statement vector is assumed to be initialized, and is
   modified as specified by TYPE.%
   %-----------------%
   PROCEDURE (stvwrk, type);
   LOCAL word; %current word in statement vector%
   REF stvwrk, word;
   IF stvwrk = 0 THEN   % treat the origin statement specially %
      CASE type OF
         =sub:
            BEGIN
            stvwrk ← 1;
            [&stvwrk + 1] ← 1;
```

```
            END;
         =suc, =pred, =up: NULL;
         ENDCASE err(0)
   ELSE
      BEGIN
      &word ← &stvwrk + stvwrk;
      CASE type OF
         =sub:
            BEGIN
            IF (stvwrk ← stvwrk + 1) > svmxlev THEN err(5);
            [&word + 1] ← 1;
            END;
         =suc:  word ← word + 1;
         =pred: word ← MAX (word-1, 1);
         =up:   stvwrk ← MAX (stvwrk-1, 0);
         ENDCASE err(0);
      END;
   RETURN;
   END.

(stpos)
   %Given a STID, this routine returns an integer which is
   the position of that statment within its plex.%
   %----------------%
   PROCEDURE (stid);
   LOCAL
      nstid,
      posit; %current position%
   IF getfhd(stid) THEN RETURN (1);
   posit ← 1;
   nstid ← gethed(stid);
   UNTIL nstid = stid DO
      BEGIN
      nstid ← getsuc(nstid);
      BUMP posit;
      END;
   RETURN (posit);
   END.


(getlev)
   %Called with STID, returns level of that statement.%
   %----------------%
   PROC(stid);
   LOCAL level; %current level%
   level ← 0;
   UNTIL stid.stpsid = origin DO
      BEGIN
      stid ← getup(stid);
      BUMP level;
      END;
   RETURN (level);
   END.


(fechno)
```

```
   %The statement number of a STID can be obtained by this
   routine.  Give the STID as the first argument, and the
   address of the A-string which is to contain the statement
   number as the second.  The statement number will be built
   in the A-string.  If the structure is not intact or the
   statement vector cannot be built, a call to RERROR or an
   EXCEED CAPICITY ERROR may result.%
   %-----------------%
   PROCEDURE(stid, astr);
   LOCAL fchsvw[50];
   stvect(stid, $fchsvw);
   fechnm($fchsvw, astr);
   RETURN;
   END.

(fechnp)
   %This is like FECHNO, but generates the statement number
   of the successor (whether it exists or not).%
   %-----------------%
   PROCEDURE(stid, astr);
   LOCAL fchsvw[50];
   stvect(stid, $fchsvw);
   stvmod($fchsvw, suc);
   fechnm($fchsvw, astr);
   RETURN;
   END.

(fechnd)
   %This is like FECHNO, but generates the statement number
   of the sub (whether it exists or not).%
   %-----------------%
   PROCEDURE(stid, astr);
   LOCAL fchsvw[50];
   stvect(stid, $fchsvw);
   stvmod($fchsvw, sub);
   fechnm($fchsvw, astr);
   RETURN;
   END.

(fechnm)
   %This routine will return the statement number, given the
   statement vector.  The address of the first word of the
   statement vector is provided as the first srgument, and
   the address of the A-string for the statement number is
   given as the second.
   The algorithm used is roughly as follows:
      The field of the statement vector corresponding to the
      current level is divided by a base.
         Two bases are used: 10 (for digits) and 27 (for
         letters and &).
      The quotient is added to an appropriate offset to create
      an ascii character, and this character is then added
      to the A-string.
      The division is repeated with the remainder until the
      quotient is less than the base. (This permits statement
      numbers of the form 12AB12D@.)
```

```
      The base is then changed and the statement vector field
      for the next level is then divided as above.%
   %-----------------%
   PROCEDURE(stvwrk, astr);
   LOCAL
      base, %current base%
      basnxt, % next base %
      curlev, %current level%
      posit, %current plex position%
      power, %power of base in plex postion = number of
         characters in statement number for this plex%
      char, %next character in statement number%
      offset, %converts number indicating position
         to number/alpha character%
      offnxt;  %next offset%
   REF astr, stvwrk;
   IF stvwrk = 0 THEN
      BEGIN
      *astr* ← *astr*, '0;
      RETURN;
      END;
   (base, basnxt, offset, offnxt) ←
      (numbase, alphbase, numoff, alphoff);
   curlev ← 0;
   LOOP
      BEGIN
      IF (curlev := curlev + 1) >= stvwrk THEN RETURN;
      posit ← stvwrk[curlev];
      %find largest power of base contained in plex position%
      power ← 1;
      WHILE power*base <= posit DO power ← power*base;
      %now put out the actual string%
      UNTIL power < 1 DO
         BEGIN
         DIV posit / power, char, posit;
         *astr* ← *astr*, char + offset;
         power ← power / base;
         END;
      (base, basnxt, offset, offnxt) ←
         (basnxt, base, offnxt, offset);
      END;
   END.


(fechux)
   %Given the address of an a-string as the first argument
   and a file number as the second, this routine returns the
   STID of the statement whose statement number is contained
   in the A-string.
   If the statement does not exist (or FECHUX is passed an
   illegal statement number) it returns a -1.
   The algorthm used is roughly as follows:
      The plex position indicated by each (alphabetic or
      numeric) level field in the statement number is converted
      to an integer, then the psid of the statement
      corresponding to that position is found.
```

```
   LODCHR is used to read characters from the statement
   number, and the resulting ascii character is converted
   to a number.
   This number is calculated by subtracting an appropriate
   offset (20B for numbers and 40B for lettrs), then
   multiplying the difference by an appropriate base (10
   for digits and 27 for letters).
   This number is added to any plex position already
   calculated.
   This subtraction, multiplication, and addition continues
   until a new level in the statement number is
   encountered.%
%-----------------%
PROCEDURE (astr, fileno);
LOCAL stid, %current stid%
   curlen, %position current character in statemet number%
   curc, %current character in statemeet number%
   base, %base for converting character to number%
   basnxt, %next base%
   zero, %zero point for converting character to number%
   zeronxt,  %next zero%
   posit; %current plex position%
REF astr;
posit ← 0;
stid ← origin;
stid.stfile ← fileno;
curlen ← 1;
base ← numbase;
basnxt ← alphbase;
zero ← numoff;
zeronxt ← alphoff;
LOOP
   BEGIN
   LOOP %read characters, calculate position%
      BEGIN
      curc ← *astr*[curlen];
      IF curc IN ['a, 'z] THEN curc ← curc - upcase;
      % check if have gone from alpha to digit or vice versa
      %
      IF (curc ← curc-zero) NOT IN [0, base) THEN EXIT;
      posit ← posit * base + curc;
      IF (curlen ← curlen+1) > astr.L THEN EXIT;
      END;
   %get stid in position indicated by posit, one level down
   from stid%
   IF posit = 0 THEN RETURN
      (IF stid.stpsid = origin THEN stid ELSE endfil);
   IF (stid := getsub(stid)) = stid THEN RETURN (endfil);
   UNTIL (posit ← posit-1) = 0 DO
      BEGIN
      IF getftl(stid) THEN RETURN (endfil);
      stid ← getsuc(stid);
      END;
   IF curlen > astr.L THEN RETURN (stid);
   (base, basnxt, zero, zeronxt, posit) ←
      (basnxt, base, zeronxt, zero, 0);
```

```
        END;
    END.


(fechsig)
    PROCEDURE (stid, astrng); %fetch statement signature%
    %assumes real statement--not an a-string%
    %----------------%
    LOCAL sdbadr, word, bytptr, count, char;
    REF sdbadr, astrng;
    % initials %
        word ← getint(stid);
        bytptr ← stbptr(empty) + $word;
        count ← 0;
        UNTIL (count ← count + 1) > 4 DO
            IF (char ← ↑bytptr) = 0 THEN EXIT LOOP
            ELSE *astrng* ← *astrng*, char;
        *astrng* ← *astrng*, SP;
    % date and time %
        lodsdb(getsdb(stid) : &sdbadr);
        <AUXCOD, dtfrmt>(sdbadr.stime, &astrng);
    RETURN;
    END.

FINISH of seqgen
```

SPLMNP

```
<NLS>SPLMNP.NLS;5, 1-JUN-71 17:52 WHP ;
FILE splmnp  % L10  <REL-NLS>SPLMNP %

   %...Statement property list procedures...%

      (copspl) PROCEDURE(source, dest);
         LOCAL ostdb, nstdb;
         IF (ostdb ← getspl(dest)) # 0 THEN deldl(ostdb);
         IF (ostdb ← getspl(source)) = 0 THEN RETURN;
         nstdb ← copdl(ostdb, dest.stfile);
         stoprop(dest, nstdb);
         RETURN END.

      (stoprop) PROCEDURE(stid, stdb);
         LOCAL b, n, ostdb, p;
         REF n, p;
         b ← lodsdb(stdb : &n);
         frzblk(b, 1);
         IF (ostdb ← getspl(stid)) = 0 THEN
            BEGIN
            stospl(stid, stdb);
            n.sback ← stid;
            n.sxflag ← FALSE;
            END
         ELSE
            LOOP
               BEGIN
               lodsdb(ostdb : &p);
               IF p.stype = n.stype THEN
                  BEGIN
                  indlrt(ostdb, stdb);
                  deldb(ostdb);
                  EXIT;
                  END;
               IF (ostdb.stpsdb ← p.sright) = 0 THEN
                  BEGIN
                  indlrt(ostdb, stdb);
                  EXIT;
                  END;
               END;
         frzblk(b, -1);
         RETURN END.

      (getprop) PROCEDURE(stid, type);
         LOCAL stdb, p;
         REF p;
         IF (stdb ← getspl(stid)) = 0 THEN RETURN(0);
         LOOP
            BEGIN
            lodsdb(stdb : &p);
            IF p.stype = type THEN RETURN(stdb);
            IF (stdb.stpsdb ← p.right) = 0 THEN RETURN(0);
            END;
         END.

      (delprop) PROCEDURE(stid, type);
```

```
        LOCAL stdb, p;
        REF p;
        IF (stdb ← getspl(stid)) = 0 THEN RETURN(0);
        LOOP
           BEGIN
           loadsdb(stdb : &p);
           IF p.stype = type THEN
              BEGIN
              deldb(stdb);
              RETURN;
              END;
           IF (stdb.stpsdb ← p.right) = 0 THEN RETURN;
           END;
        END.

%...Data list procedures...%

    (copdl) PROCEDURE(stdb, fileno);
        LOCAL nstdb, down, newdown, right, newright;
        nstdb ← copsdb(stdb, fileno);
        IF down ← getxdown(stdb) THEN
           BEGIN
           newdown ← copdl(down, fileno);
           stoxdown(nstdb, newdown);
           stoxback(newdown, nstdb);
           END;
        IF right ← getxright(stdb) THEN
           BEGIN
           newright ← copdl(right, fileno);
           stoxright(nstdb, newright);
           stoxback(newright, nstdb);
           END;
        RETURN(nstdb) END.

    (indldn) PROCEDURE(old, new);
        LOCAL down, tail, next;
        IF (down ← getxdown(old)) # 0 THEN
           BEGIN
           tail ← new;
           UNTIL (next ← getxright(tail)) = 0 DO tail ← next;
           stoxback(down, tail);
           stoxright(tail, down);
           END;
        stoxdown(old, new);
        stoxback(new, old);
        RETURN END.

    (indlrt) PROCEDURE(old, new);
        LOCAL right, tail, next;
        IF (right ← getxright(old)) # 0 THEN
           BEGIN
           tail ← new;
           UNTIL (next ← getxright(tail)) = 0 DO tail ← next;
           stoxback(right, tail);
           stoxright(tail, right);
           END;
```

```
      stoxright(old, new);
      stoxback(new, old);
      RETURN END.

   (deldl) PROCEDURE(stdb);
      LOCAL b, p, right;
      REF p;
      b ← lodsdb(stdb : &p);
      frzblk(b, 1);
      right ← stdb;
      WHILE (right.stpsdb ← p.sright) # 0 DO deldb(right);
      deldb(stdb);
      frzblk(b, -1);
      RETURN END.

%...Data branch procedures...%

   (copdb) PROCEDURE(stdb, fileno);
      LOCAL nstdb, down, ndown;
      nstdb ← copsdb(stdb, fileno);
      IF (down ← getxdown(stdb)) # 0 THEN
         BEGIN
         ndown ← copdl(down, fileno);
         indldn(nstdb, ndown);
         END;
      RETURN(nstdb) END.

   (remdb) PROCEDURE(stdb);
      LOCAL b, sdb, right, p;
      REF sdb, p;
      b ← lodsdb(stdb : &sdb);
      frzblk(b, 1);
      right ← stdb;
      IF (right.stpsdb ← sdb.sright) # 0 THEN
         BEGIN
         lodsdb(right : &p);
         p.sback ← sdb.sback;
         p.sxflag ← sdb.sxflag;
         END;
      IF sdb.sxflag THEN
         BEGIN
         stdb.stpsdb ← sdb.sback;
         stoxright(stdb, right);
         END
      ELSE
         stospl(makstid(stdb.stfile, sdb.sback), right);
      sdb.sback ← sdb.sright ← 0;
      frzblk(b, -1);
      RETURN END.

   (deldb) PROCEDURE(stdb);
      LOCAL down %etc% ;
      IF (down ← getxdown(stdb)) # 0 THEN deldl(down);
      remdb(stdb);
      % delete and consolidate as in fresdb now %
      END.
```

```
%...Statement data block procedures...%

    (copsdb) PROCEDURE(stdb, fileno);
       LOCAL n;
       REF n;
       %... make copy like now ...%
       n.sback ← n.sright ← n.sdown ← 0;
       n.sxflag ← TRUE;
       END.

FINISH of splmnp

    in gcol
    nstdb ← stdb;
    IF (nstdb.stpsdb ← p.sright) # 0 THEN
       stoxback(nstdb, new);
    IF (nstdb.stpsdb ← p.sdown) # 0 THEN
       stoxback(nstdb, new);
    IF p.sxflag THEN
       BEGIN
       nstdb.stpsdb ← p.sback;
       lodsdb(nstdb : &q);
       CASE old.stpsdb OF
          = q.sdown  : q.sdown ← new;
          = q.sright : q.sright ← new;
          ENDCASE badfil(fileno);
       END;
    ELSE stospl(makstid(fileno, p.sback), new);
```

STR MNP

```
<NLS>STRMNP.NLS;18, 4-NOV-71 15:22 WHP ;
FILE strmnp    %   L10   <REL-NLS>STRMNP     %

vdeclarations%

   DECLARE  suc=1, sub=0;
   DECLARE  ctoff=FALSE, cton=TRUE;

%append statement routines%

   (apchk)
      %This routine checks that legal items have been chosen for append
      statement.  A statement cannot be appended to itself, nor can a
      statement be appended to a part of its own substructure.%
      %-------------%
      PROC(target, source);
      LOCAL srsub;
      IF target = source THEN err($"illegal append");
      IF (srsub ← getsub(source)) # source THEN
         movtst( target, srsub, getail(srsub));
      RETURN;
      END.

   (apdo)
      %This routine is called during the append statement process. It
      deletes the second statement specified after the text in the
      statement has been appended to the first.%
      %-------------%
      PROC(target, source, astring);
      LOCAL srsub;
      apchk(target, source);
      IF (srsub ← getsub(source)) # source THEN
         BEGIN
         *ladj* ← 'D;
         movplex( target, srsub, $ladj);
         END;
      <TXTEDT, staptx>(target, source, astring);
      delbranch(source);
      RETURN;
      END.

%break statement%

   (bkstat)
      %This routine gets a new stid, inserts it after OLDSTD, in the
      direction specified by DIR.  It then creates a new sdb for the
      statement using the tpointer and astring passed it.  It returns
      the value of the new stid it obtained.%
      %-------------%
      PROCEDURE(tptr, levstg, astring);
      LOCAL newstd; %new stid%
      REF tptr;
      newstd ← newrng(tptr.stfile);
      insgrp(tptr, newstd, newstd, levstg);
      <TXTEDT, brkst>(&tptr, newstd, astring);
      RETURN(newstd);
```

```
    END.

%primary control for structure manipulation%

    %copy%

        (copstat) PROC(target, source, levstg);
            LOCAL newsrc; %new stid%
            newsrc ← newrng(target.stfile);
            insgrp(target, newsrc, newsrc, levstg);
            copsdb(source, newsrc);
            RETURN(newsrc);
            END.

        (copbranch) PROCEDURE(target, source, levstg);
            LOCAL newsrc; %new stid%
            newsrc ← copgrp(target, source, source, levstg, ctoff);
            RETURN(newsrc);
            END.

        (copgroup) PROCEDURE(target, src1, src2, levstg);
            LOCAL newsrc; %new stid%
            src1 ← grptst(src1, src2 : src2);
            newsrc ← copgrp(target, src1, src2, levstg, ctoff);
            RETURN(newsrc);
            END.

        (copplex) PROCEDURE(target, source, levstg);
            LOCAL src1, %beginning of plex%
                src2, %end of plex%
                newsrc; %new stid%
            src1 ← plxset(source : src2);
            newsrc ← copgrp(target, src1, src2, levstg, ctoff);
            RETURN(newsrc);
            END.

    %delete%

        (xdele)
            %Given two stid's, this routine will delete the group bounded
            by them.  It deletes it from the ring, using REMGRP, and frees
            its SDB's and ring elements, using DELGRP.  It assumes that
            STID1, STID2 define a legal, ordered group.%
            %------------%
            PROCEDURE(stid1, stid2);
            LOCAL newsid;
            newsid ← getsuc(stid2);
            IF NOT remgrp(stid1, stid2) THEN err($"illegal delete");
            delgrp(stid1, stid2, newsid);
            RETURN;
            END.

        (delstat) PROCEDURE(stid);
            IF getsub(stid) # stid THEN err($"illegal delete");
            xdele(stid, stid);
            RETURN;
```

```
    END.

(delbranch) PROCEDURE(stid);
    xdele(stid, stid);
    RETURN;
    END.

(delgroup) PROCEDURE(stid1, stid2);
    stid1 ← grptst(stid1, stid2 :stid2);
    xdele(stid1, stid2);
    RETURN;
    END.

(delplex) PROCEDURE(stid);
    LOCAL stid1, %beginning of plex%
        stid2; %end of plex%
    stid1 ← plxset(stid :stid2);
    xdele(stid1, stid2);
    RETURN;
    END.

%insert%

(instat)
    %Given an stid, an astring, and a levstgection, this routine
    will insert the text in the astring as a statement after the
    stid passed, in the levstgection specified.  It returns the
    value of the new statement's stid.%
    %--------------%
    PROCEDURE(target, astring, levstg);
    REF astring;
    RETURN(inslit(target, &astring, levstg));
    END.

%move%

(xmove)
    %This routine assumes that SRC1 and SRC2 define the bounds of a
    legal, ordered group to be moved and that TARGET contains the
    target stid.  It checks that the target is not contained within
    the group.  If not, it removes the group from the structure, by
    calling REMGRP, and inserts it after TARGET, by calling
    INSGRP.%
    %--------------%
    PROCEDURE(target, src1, src2, levstg);
    LOCAL newsrc; %new stid (end of group moved)%
    movtst(target, src1, src2);
    IF target.stfile = src1.stfile THEN
        BEGIN %intrafile move%
        IF NOT remgrp(src1, src2) THEN err($"illegal move");
        insgrp(target, src1, src2, levstg);
        newsrc ← src2;
        END
    ELSE
        BEGIN %cross file move%
        newsrc ← copgrp(target, src1, src2, levstg, cton);
```

```
      xdele(src1, src2);
      END;
   RETURN(newsrc);
   END.

(movstat) PROCEDURE(target, source, levstg);
   IF getsub(source) # source THEN err($"illegal move");
   RETURN( xmove(target, source, source, levstg));
   END.

(movbranch) PROCEDURE(target, source, levstg);
   RETURN( xmove(target, source, source, levstg));
   END.

(movgroup) PROCEDURE(target, src1, src2, levstg);
   src1 ← grptst(src1, src2 : src2);
   RETURN( xmove(target, src1, src2, levstg));
   END.

(movplex) PROCEDURE(target, source, levstg);
   LOCAL src1, %beginning of plex%
      src2; %end of plex%
   src1 ← plxset(source :src2);
   RETURN( xmove(target, src1, src2, levstg));
   END.

%replace%

(xrepl)
   %This routine replaces the group delimited by TGT1,2 by that
   delimited by SRC1,2.  First it copies SRC1,2 after TGT1,2; then
   it deletes TGT1,2.  (It assumes that the stid's define legal,
   ordered groups.)%
   %------------%
   PROCEDURE(tgt1, tgt2, src1, src2);
   LOCAL newsrc; %new stid (end of group)%
   newsrc ← copgrp(tgt2, src1, src2, sucdir, ctoff);
   xdele(tgt1, tgt2);
   RETURN(newsrc);
   END.

(repstat) PROCEDURE(bugdit, target, source, astring);
   REF astring;
   dsttx(target); %free SDB%
   IF bugdit THEN
      copsdb(source, target)
   ELSE %literal input%
      ST target ← *astring*;
   RETURN(target);
   END.

(repbranch) PROCEDURE(bugdit, target, source, astring);
   IF bugdit THEN
      RETURN ( xrepl (target, target, source, source))
   ELSE RETURN ( rpllit (target, target, astring));
   END.
```

```
(repgroup)
   PROCEDURE(bugdit, tgt1, tgt2, src1, src2, astring);
   tgt1 ← grptst(tgt1, tgt2 :tgt2);
   IF bugdit THEN
      BEGIN
      src1 ← grptst(src1, src2 :src2);
      RETURN( xrepl(tgt1, tgt2, src1, src2));
      END
   ELSE RETURN(rpllit(tgt1, tgt2, astring));
   END.

(repplex) PROCEDURE(bugdit, target, source, astring);
   LOCAL
      src1, %beginning of plex to replace tgt1%
      src2, %end of plex to replace tgt2%
      tgt1, %beginning of plex to be replaced%
      tgt2; %end of plex to be replaced%
   tgt1 ← plxset(target : tgt2);
   IF bugdit THEN
      BEGIN
      src1 ← plxset(source : src2);
      RETURN( xrepl(tgt1, tgt2, src1, src2));
      END
   ELSE RETURN(rpllit(tgt1, tgt2, astring));
   END.

%transpose%

(xtran)
   %This routine transposes the groups defined by FIRST1 and
   FIRST2 (first group) and SECND1 and SECND2 (second group).  The
   first group need not be ahead of the second in the file
   structure, but the routine assumes that FIRST1,2 and SECND1,2
   define legal groups.  It checks to see that the groups don't
   overlap.
      If one group follows the other immediately, then the second
      group is merely deleted from the structure and inserted
      after the pred (or source) of the first; the first the is in
      the right position automatically.  Otherwise, the routine
      finds the pred (or source) of each group.
         If the two groups are in the same file, it calls REMGRP
         to delete one group from the structure, then INSGRP to
         insert the group after the pred (or source) of the other
         group.  The same delete/insert sequence is followed for
         the other group.  Otherwise, it uses COPGROUP twice and
         XDELE twice to perform the requisite operations.%
   %-------------%
   PROCEDURE(first1, first2, secnd1, secnd2);
   LOCAL
      gp1tgt, %target for group delimited by first1,first2%
      gp1dir, %direction for insertion of above group%
      gp2tgt, %target for group delimited by secnd1,secnd2%
      gp2dir; %direction for insertion of above group%
   trntst(first1, first2, secnd1, secnd2);
   IF getfhd(secnd1) THEN
```

```
        BEGIN
        gpltgt ← getup(secnd2);
        gpldir ← down;
        END
    ELSE
        BEGIN
        gpldir ← sucdir;
        IF (gpltgt ← getprd(secnd1)) = first2 THEN
            BEGIN %SECND1,2 follows FIRST1,2 directly%
            IF NOT remgrp(first1, first2) THEN
                err($"illegal transpose");
            insgrp(secnd2, first1, first2, sucdir);
            RETURN;
            END;
        END;
    IF getfhd(first1) THEN
        BEGIN
        gp2tgt ← getup(first2);
        gp2dir ← down;
        END
    ELSE
        BEGIN
        gp2dir ← sucdir;
        IF (gp2tgt ← getprd(first1)) = secnd2 THEN
            BEGIN %FIRST1,2 follows SECND1,2 directly%
            IF NOT remgrp(secnd1, secnd2) THEN
                err($"illegal transpose");
            insgrp(first2, secnd1, secnd2, gp2dir);
            RETURN;
            END;
        END;
    IF first1.stfile = secnd1.stfile THEN
        BEGIN %intrafile transpose%
        IF NOT remgrp(secnd1, secnd2) THEN
            err($"illegal transpose");
        insgrp(gp2tgt, secnd1, secnd2, gp2dir);
        IF NOT remgrp(first1, first2) THEN
            err($"illegal transpose");
        insgrp(gpltgt, first1, first2, gpldir);
        END
    ELSE
        BEGIN %cross file transpose%
        copgrp(gpltgt, first1, first2, gpldir, cton);
        copgrp(gp2tgt, secnd1, secnd2, gp2dir, cton);
        xdele(first1, first2);
        xdele(secnd1, secnd2);
        END;
    RETURN;
    END.

(trntst)
    %This routine checks that the groups specified by the transpose
    commands have legal bounds.  It does this by calling MOVTST to
    see if FIRST1,2 are in the group bounded by SECND1,2 or
    SECND1,2 in the group bounded by FIRST1,2.%
    %------------%
```

```
      PROCEDURE(first1, first2, secnd1, secnd2);
      movtst(first1, secnd1, secnd2);
      movtst(first2, secnd1, secnd2);
      movtst(secnd1, first1, first2);
      movtst(secnd2, first1, first2);
      RETURN;
      END.

   (trstat)
      PROCEDURE(stid1, stid2);
      LOCAL
         psdb1, %psdb of stid1%
         psdb2, %psdb of stid2%
         newsd1, %new stid, replaces stid1 in cross-file
         trpose%
         newsd2, %new stid, replaces stid2 in cross-file
         trpose%
         tmpstd, %temporary to save stid%
         sdbadr; %adress of sdb%
      REF sdbadr;
      IF stid1.stfile = stid2.stfile THEN
         BEGIN  %intrafile transpose%
         psdb1 ← getsdb(stid1);  psdb2 ← getsdb(stid2);
         lodsdb(psdb1 :&sdbadr); sdbadr.spsid ← stid2.stpsid;
         stosdb(stid2, psdb1);
         lodsdb(psdb2 :&sdbadr); sdbadr.spsid ← stid1.stpsid;
         stosdb(stid1, psdb2);
         END
      ELSE
         BEGIN %cross file transpose%
         newsd1 ← copstat(stid1, stid2, sucdir);
         upctbl(stid2, newsd1, stid2);
         newsd2 ← copstat(stid2, stid1, sucdir);
         upctbl(stid1, newsd2, stid1);
         IF (tmpstd ← getsub(stid1)) # stid1 THEN
            movplex(newsd1, tmpstd, down);
         xdele(stid1, stid1);
         IF (tmpstd ← getsub(stid2)) # stid2 THEN
            movplex(newsd2, tmpstd, down);
         xdele(stid2, stid2);
         END;
      RETURN;
      END.

   (trbranch) PROCEDURE(first, second);
      xtran(first, first, second, second);
      RETURN;
      END.

   (trgroup) PROCEDURE(first1, first2, secnd1, secnd2);
      first1 ← grptst(first1, first2 : first2);
      secnd1 ← grptst(secnd1, secnd2 : secnd2);
      xtran(first1, first2, secnd1, secnd2);
      RETURN;
      END.
```

```
   (trplex) PROCEDURE(first, second);
      LOCAL
         first1, %beginning of first plex%
         first2, %end of first plex%
         secnd1, %beginning of second plex%
         secnd2; %end of second plex%
      first1 ← plxset(first : first2);
      secnd1 ← plxset(second : secnd2);
      xtran(first1, first2, secnd1, secnd2);
      RETURN;
      END.

%xset%

   (xstruc)
      %Given two stid's, this routine will XSET each statement in the
      group bounded by them.  It assumes that STID1, STID2 define a
      legal, ordered group.%
      %-------------%
      PROCEDURE(stid1, stid2);
      grpapply(stid1, stid2, $cshftall);
      RETURN;
      END.

   (xsetbranch) PROCEDURE(stid);
      xstruc(stid, stid);
      RETURN;
      END.

   (xsetgroup) PROCEDURE(stid1, stid2);
      stid1 ← grptst(stid1, stid2 :stid2);
      xstruc(stid1, stid2);
      RETURN;
      END.

   (xsetplex) PROCEDURE(stid);
      LOCAL stid1, %beginning of plex%
         stid2; %end of plex%
      stid1 ← plxset(stid :stid2);
      xstruc(stid1, stid2);
      RETURN;
      END.

%assimilate%

   (xassim) PROCEDURE
      (target, src1, src2, levstg, vspec1, vspec2, usqcod, cacode);
      LOCAL newtgt, newsrc, lstlev, toplev, sqgwrk;
      REF levstg, sqgwrk;
      &sqgwrk ← openseq (src1, src2, vspec1, vspec2, usqcod, cacode);
      IF (newsrc ← seqgen(&sqgwrk)) # endfil THEN
         BEGIN
         newtgt ←
            IF newsrc.stastr THEN
               instat (target, newsrc.stadr, &levstg)
            ELSE copstat(target, newsrc, &levstg);
```

```
            toplev ← lstlev ←
               IF sqgwrk.swclvl = 0 THEN 1 ELSE sqgwrk.swclvl;
            WHILE (newsrc ← seqgen(&sqgwrk)) # endfil DO
               BEGIN
               *levstg* ← NULL;
               CASE sqgwrk.swclvl OF
                  =lstlev: NULL;
                  >lstlev:
                     BEGIN
                     *levstg* ← 'D;
                     lstlev ← lstlev + 1;
                     END;
                  <lstlev:
                     WHILE lstlev > toplev DO
                        BEGIN
                        *levstg* ← *levstg*, 'U;
                        IF ((lstlev ← lstlev - 1) = sqgwrk.swclvl) THEN
                           EXIT LOOP 1;
                        END;
                  ENDCASE NULL;
               newtgt ←
                  IF newsrc.stastr THEN
                     instat (newtgt, newsrc.stadr, &levstg)
                  ELSE copstat (newtgt, newsrc, &levstg);
               END;
            END;
         closeseq (&sqgwrk);
         RETURN;
         END.

%structural inserts%

   (newsuc)
      %This routine gets a new stid and then inserts it as the suc of
      the stid passed it.  It returns the new stid.%
      %-------------%
      PROCEDURE(stid);
      LOCAL newstd; %new stid%
      newstd ← newrng(stid.stfile);
      inss(stid, newstd, newstd);
      RETURN(newstd);
      END.

   (newsub)
      %This routine gets a new stid and then inserts it as the sub of
      the stid passed it.  It returns the new stid.%
      %-------------%
      PROCEDURE(stid);
      LOCAL newstd; %new stid%
      newstd ← newrng(stid.stfile);
      insd(stid, newstd, newstd);
      RETURN(newstd);
      END.

   (insgrp)
      %Insert SRC1, SRC2 at TARGET according to DIR.  If DIR is true,
```

```
   then the group is inserted as the successor; otherwise it is
   inserted as the sub.  (SRC1 and SRC2 are assumed to define a
   legal, ordered group.)%
   %-------------%
   PROCEDURE(target, src1, src2, levstg);
   LOCAL dir;
   REF levstg;
   IF levstg.L # empty THEN target ← levset(target, &levstg : dir)
   ELSE dir ← suc;
   IF target.stpsid # origin AND dir THEN
      inss(target, src1, src2)
   ELSE insd(target, src1, src2);
   RETURN;
   END.

(inss)
   %Given an stid, this routine inserts a group, defined by the
   second and third arguments, as the suc of the first argument.
   First it makes the suc of STID the suc of GRP2, and updates the
   tail flag; then it makes GRP1 the suc of STID and updates the head
   and tail flags.%
   %-------------%
   PROCEDURE(stid, grp1, grp2);
   IF stid.stfile # grp1.stfile THEN err($"illegal insert");
   IF grp1.stfile # grp2.stfile THEN err($"illegal group");
   stosuc(grp2, getsuc(stid));
   stoftl(grp2, getftl(stid));
   stosuc(stid, grp1);
   stofhd(grp1, FALSE);
   stoftl(stid, FALSE);
   RETURN;
   END.

(insd)
   %Given an stid, this routine inserts the group defined by the
   second and third arguments down from the first argument.%
   %-------------%
   PROCEDURE(stid, grp1, grp2);
   LOCAL substd; %stid of sub of stid passed as argument%
   IF stid.stfile # grp1.stfile THEN err($"illegal insert");
   IF grp1.stfile # grp2.stfile THEN err($"illegal group");
   IF (substd ← getsub(stid)) # stid THEN
      BEGIN
      stofhd(substd, FALSE);
      stoftl(grp2, FALSE);
      END
   ELSE stoftl(grp2, TRUE);
   stosuc(grp2, substd);
   stosub(stid, grp1);
   stofhd(grp1, TRUE);
   RETURN;
   END.

(levset)
   %Combines STID and DIR to give final target psid for level
   specification during structural editing.  If the new statement is
```

```
      to be the successor of the stid returned, DIR will be TRUE upon
      completion of this routine, otherwise the new statement will be
      inserted as the down of the stid returned.%
      %---------------%
      PROCEDURE(stid, levstg);
      LOCAL dir, count, numb;
      REF levstg;
      dir ← 0;
      IF levstg.L # empty THEN
         BEGIN
         count ← 1;
         DO
            CASE *levstg*[count] OF
               IN ['0, '9]:
                  BEGIN
                  numb ← *levstg*[count] - '0;
                  BUMP count;
                  CASE *levstg*[count] OF
                     ='d, ='D: dir ← dir + numb;
                     ='u, ='U, =ENDCHR: dir ← dir - numb;
                     ENDCASE;
                  END;
               ='U, ='u: BUMP DOWN dir;
               ='D, ='d: BUMP dir;
               ENDCASE NULL
         UNTIL (count ← count + 1) > levstg.L;
         END;
      CASE dir OF
         =0: BUMP dir; %successor%
         >0: dir ← 0 %down%
         ENDCASE %up number of levels indicated by dir%
      WHILE (dir ← dir + 1) <= 0
         DO stid ← <STR10, getup>(stid);

      RETURN(stid, dir);
      END.

%insert and replace using literal register%

   (inslit)
      %This routine gets a new stid, inserts it after the stid passed
      it, in the direction specified, and copies the A-string passed it
      into the statement's SDB.  It returns the value of the new stid.%
      %-------------%
      PROCEDURE(target, astring, dir);
      LOCAL newstid; %new stid%
      REF astring;
      newstid ← newrng(target.stfile);
      insgrp(target, newstid, newstid, dir);
      ST newstid ← *astring*;
      RETURN(newstid);
      END.

   (rpllit)
      %This routine replaces the group defined by the first two
      arguments passed it with a new statement containing the text in
```

```
      the astring passed it.  It returns the value of the new stid.
         First it gets a new stid, and inserts it as the successor of
         GRP2.  It then deletes the group bounded by GRP1, GRP2, and
         puts the text in the astring passed it in the new SDB.%
      %-------------%
      PROCEDURE(grp1, grp2, astring);
      LOCAL stid; %stid for new statement%
      REF astring;
      IF grp1.stfile # grp2.stfile THEN err($"illegal group");
      stid ← newrng(grp1.stfile);
      inss(grp2, stid, stid);
      xdele(grp1, grp2);
      ST stid ← *astring*;
      RETURN(stid);
      END.

%test and set bounds of structure%

   (plxset)
      %Given a stid, this routine will return the stid's of the head and
      tail, respectively, of the plex in which the stid appears.%
      %-------------%
      PROCEDURE(stid);
      LOCAL grp1, %stid of head%
         grp2; %stid of tail%
      IF stid.stpsid = origin THEN grp1 ← grp2 ← stid
      ELSE
         BEGIN
         grp1 ← gethed(stid);
         grp2 ← getail(stid);
         END;
      RETURN(grp1, grp2);
      END.

   (grptst)
      %Given two stid's, this routine checks that they specify a legal
      group; it also returns them ordered (GRP1,GRP2). If the stid's do
      not form a legal group, an err($"illegal group") is issued.%
      %-------------%
      PROCEDURE(stid1, stid2);
      LOCAL t1, %working stid1%
         t2; %working stid2%
      IF stid1.stpsid = origin OR stid2.stpsid = origin
      OR stid1.stfile # stid2.stfile THEN err($"illegal group");
      t1 ← stid1; t2 ← stid2;
      LOOP
         BEGIN %is stid2 on same level, and after, stid1%
         IF t1 = stid2 THEN RETURN(stid1, stid2);
         IF getft1(t1) THEN LOOP
            BEGIN %is stid1 on same level after stid2%
            IF t2 = stid1 THEN RETURN(stid2, stid1);
            IF getft1(t2) THEN err($"illegal group");
            t2 ← getsuc(t2);
            END;
         t1 ← getsuc(t1);
         END;
```

```
    END.

(movtst)
    %Given an stid as the first argument, this routine makes sure that
    the stid is not in the group bounded by the second and third
    arguments passed it.  If it is, it does an err.
       First it sees whether STID is a suc of GRP2 on the same level.
       If not, it then sees whether STID's sucs (and ups) are GRP1 or
       GRP2.%
    %-------------%
    PROCEDURE(stid, grp1, grp2);
    LOCAL tmpsid; %working stid%
    IF grp2.stpsid = origin THEN err($"illegal move");
    IF grp1.stfile # grp2.stfile THEN err($"illegal group");
    IF stid.stfile # grp1.stfile
       OR stid.stpsid = origin THEN RETURN;
    IF stid = grp1 OR stid = grp2 THEN err($"illegal move");
    tmpsid ← grp2;
    LOOP %is stid in plex after grp2%
       BEGIN
       IF getft1(tmpsid) THEN
          BEGIN
          tmpsid ← getsuc(tmpsid);
          EXIT;
          END;
       tmpsid ← getsuc(tmpsid);
       IF stid = tmpsid THEN RETURN; %psid in plex after grp2%
       END;
    LOOP %see if grp1, grp2 in superstructure of psid%
       BEGIN
       IF stid = grp1 THEN err($"illegal move");
       IF stid.stpsid = origin OR stid = tmpsid THEN RETURN;
       LOOP
           BEGIN
           IF stid = grp2 THEN err($"illegal move");
           IF getft1(stid) THEN EXIT;
           stid ← getsuc(stid);
           IF stid = grp1 THEN RETURN;
           END;
        stid ← getsuc(stid);
        END;
    END.

%basic manipulation routines%

    (grpapply)
       %Assumes that stid1 and stid2 form a legal group with stid1 the
       first statement and stid2 the second.  For each statement in the
       group, calls proc(stid).%
       %-----------------%
       PROCEDURE(stid1, stid2, proc);
       LOCAL stid;  REF proc;
       stid ← stid1;
       LOOP
          BEGIN
          proc(stid);
```

```
      % go down the tree %
      WHILE (stid := getsub(stid)) # stid DO proc(stid);
      % have reached a statement with no substructure %
      LOOP
         BEGIN
         % at this point have applied proc to stid and all of stid's
         substructure %
         IF stid = stid2 THEN RETURN;
         IF stid = stid1 THEN
            BEGIN
            stid1 ← stid ← getsuc(stid);
            EXIT LOOP;
            END;
         IF NOT getft1(stid := getsuc(stid)) THEN EXIT LOOP;
         END;
      END;
   END.


(copgrp)
   %Given an stid as the first argument, this routine copies the
   group bounded by the second and third arguments after the first
   stid, in the direction specified by the fourth argument.  The
   fifth argument indicates whether the correspondence list should be
   updated.
      It proceeds by constructing new ring elements for each branch
      defined by the top-level statements in the group. Then it
      copies these data blocks into freshly allocated SDB's.  It then
      inserts the newly created group after the stid passed it.%
   %-------------%
   PROCEDURE(stid, grp1, grp2, dir, upctf);
   LOCAL newsid, %new stid%
      oldsid, %old stid being processed%
      newgp1; %head of new group%
   IF grp1.stfile # grp2.stfile THEN err($"illegal group");
   oldsid ← grp1;
   newsid ← newgp1 ← newrng(stid.stfile);
   LOOP
      BEGIN
      %get stid's, this branch%
      WHILE (oldsid := getsub(oldsid)) # oldsid DO
         newsid ← newsub(newsid);
      %now copy sdb's and point to next branch%
      LOOP
         BEGIN
         copsdb(oldsid, newsid);
         IF upctf THEN upctbl(oldsid, newsid, oldsid);
         IF oldsid = grp1 THEN %branch copy complete%
            BEGIN
            IF oldsid = grp2 THEN %group copy complete%
               BEGIN
               insgrp(stid, newgp1, newsid, dir);
               RETURN(newsid);
               END;
            oldsid ← grp1 ← getsuc(oldsid);
            EXIT;
```

```
            END;
         IF NOT getftl(oldsid := getsuc(oldsid)) THEN EXIT;
            %still more in plex%
         newsid ← getsuc(newsid);
         END;
      newsid ← newsuc(newsid);
      END;
   END.

(remgrp)
   %Removes group whose bounds are passed it from position in ring
   but does not delete either ring elements or SDB's.%
   %-------------%
   PROCEDURE(grp1, grp2);
   LOCAL stid; %work area for updating stid%
   IF grp1.stpsid = origin OR
      grp2.stpsid = origin THEN RETURN(FALSE);
   IF grp1.stfile # grp2.stfile THEN err($"illegal group");
   %putcdpanic(IF getfhd(grp1) THEN getup(grp1)
      ELSE getprd(grp1));%
   IF getfhd(grp1) THEN
      BEGIN
      stid ← getsuc(grp2); %new plex head%
      IF NOT getftl(grp2) THEN stofhd(stid, TRUE);
      stosub(getup(grp2), stid);
      END
   ELSE
      BEGIN
      stid ← getprd(grp1);
      stoftl(stid, getftl(grp2));
      stosuc(stid, getsuc(grp2));
      END;
   RETURN;
   END.

(delgrp)
   %This routine destroys all trace of the group bounded by the
   arguments passed it.  (Note that it does not remove the group from
   the ring; it assumes that this has been done by, for example,
   REMGRP.)
      It follows each branch to its deepest level, deleting
      references to substructure in the sub fields; it then follows
      the structure back up, through the suc pointers, freeing SDB's
      as it goes.%
   %-------------%
   PROCEDURE(grp1, grp2, newsid);
   LOCAL
      stid, %stid being processed%
      subsid; %sub of stid being processed%
   IF grp1.stfile # grp2.stfile THEN err($"illegal group");
   IF grp1.stpsid = origin OR
      grp2.stpsid = origin THEN err($"illegal delete");
   stid ← grp1;
   LOOP
      BEGIN
      WHILE (subsid ← getsub(stid)) # stid DO
```

```
            BEGIN
            stosub(stid, stid);
            stid ← subsid;
            END;
         stid ← getsuc(subsid); %now go back up%
         upctbl(subsid, endfil, newsid);
         <TXTEDT,dsttx> (subsid); %free sdb%
         frerng(subsid);
         IF subsid = grp2 THEN RETURN;
         END;
      END.

%move around in ring%

   (getail)
      %Given an stid, this procedure returns the stid of the tail of the
      current plex%
      %-------------%
      PROCEDURE (stid);
      IF stid.stpsid # origin THEN
         UNTIL getftl(stid) DO
         stid ← getsuc(stid);
      RETURN(stid);
      END.

   (getup)
      %Given an stid, this routine returns the stid of the source of the
      original stid%
      %-------------%
      PROCEDURE(stid);
      IF stid.stpsid # origin THEN
         stid ← getsuc(getail(stid));
      RETURN(stid);
      END.

   (gethed)
      %Given an stid, this routine returns the head of the current plex;
      if it is passed the origin statement, it returns the origin%
      %-------------%
      PROCEDURE(stid);
      IF stid.stpsid # origin THEN
         stid ← getsub(getup(stid));
      RETURN(stid);
      END.

   (getprd)
      %Given an stid, this routine returns the predecessor; if the psid
      heads a plex, the stid itself is returned%
      %-------------%
      PROCEDURE (stid);
      LOCAL presid, %stid of predecessor%
         sucsid; %stid of successor of presid%
      IF getfhd(stid) THEN RETURN(stid);
      presid ← gethed(stid);
      UNTIL (sucsid ← getsuc(presid)) = stid DO
         presid ← sucsid;
```

```
   RETURN(presid);
   END.

(getend)
   %This procedure returns the end of the tail of the stid passed
   it.%
   %-------------%
   PROCEDURE(stid);
   DO stid ← getail(stid)
   UNTIL (stid := getsub(stid)) = stid;
   RETURN(stid);
   END.

(getbck)
   %This procedure finds the back of the stid of the statemment
   passed it.  It does not observe viewspecs.%
   %-------------%
   PROCEDURE(stid);
   IF stid.stpsid = origin THEN RETURN(stid);
   IF getfhd(stid) THEN RETURN(getup(stid));
   stid ← getprd(stid);
   IF (stid := getsub(stid)) = stid THEN RETURN(stid);
   RETURN(getend(stid));
   END.

(getnxt)
   %This procedure finds the "next" statement, i.e. the sub,
   suc, father, grandfather, etc, of the STID passed as arg
   1.  It ignores all viewspec parameters.%
   %-----------------%
   PROCEDURE (stid);
   IF (stid := getsub(stid)) = stid THEN
      % no substructure %
      BEGIN
      LOOP
         BEGIN
         IF stid.stpsid = origin THEN RETURN (endfil);
         IF getftl(stid) = 0 THEN EXIT; % not a tail %
         stid ← getsuc(stid);
         END;
      stid ← getsuc(stid);
      END;
   IF stid.stpsid = origin THEN RETURN (endfil);
   RETURN (stid);
   END.


%correspondence table manipulation%

   (upctbl)
      %Given three stid's, this routine will update occurrences of the
      first stid, using rplsid as the stid that contains the text
      corresponding to oldsid, and newsid as the stid that is the
      replacement for oldsid.  (rplsid should be endfil if the original
      text has been eliminated.)%
      %-------------%
```

```
PROCEDURE(oldsid, rplsid, newsid);
LOCAL list, listnd;
REF list;
IF clstad = 0 OR [clstad].clbuff = 0 THEN RETURN;
&list ← [clstad].clbuff;
listnd ← &list + [clstad].clcnt * cll;
FOR &list UP cll UNTIL >= listnd
DO
   IF list.clst1 = oldsid THEN
      BEGIN
      list.clst2 ← newsid;
      list.clst1 ← rplsid;
      END
   ELSE
      IF list.clst2 = oldsid THEN
         list.clst2 ← newsid;
RETURN;
END.
```

FINISH of strmnp

SUB ST

```
<NLS>SUBST.NLS;39, 10-NOV-71 9:42 MSC ;
FILE subst  %  L10  <REL-NLS>subst  %
%.....Fast substitute for NLS, 29-OCT-71 LPD.....%
%.....Documentation.....%
   %The test strings are sorted by initial character and chained
   together, with the head of the chain in a character-code indexed
   array subdsp.  This allows a very fast check whether a character can
   possibly begin a test string.  Each test-replacement pair is
   represented by a record (card) which is linked to others for the same
   initial test character through the canxt field.%
   %Global variables used:
      subcnt  count of substitutions
      lit     A-string for building up new statement
      swork   internal work area, see (stbpget)
   %
%.....Declarations.....%
   REGISTER a1=12, a2=13;
      %*** see main loop of substat ***%
%.....Common code for DNLS and TNLS.....%
   (substitute) PROCEDURE (type, stid1, stid2, sb, da);
   %substitute%
      %process the structure whose type and stid's are passed%
      %----------%
      subcnt ← 0;   % count of number of visibles %
      CASE type OF
         = stmtv: xsubs(stid1, sb, da);
         = brnchv: xsubb(stid1, sb, da);
         = plexv: xsubp(stid1, sb, da);
         = groupv: xsubg(stid1, stid2, sb, da);
         ENDCASE err(0);
      RETURN;
      END.

   (sbinit) PROCEDURE(sbhed, sbrec, sbdsp, sbastr);
      % initialize substitute data structure %
      LOCAL j;
      POINTER sbhed;
      sbhed.sbrp ← sbrec;
      sbhed.sbdp ← sbdsp;
      sbhed.sbas ← sbastr;
      *[sbastr]* ← NULL;
      FOR j ← 0 UP UNTIL >=200B DO [sbdsp+j] ← 0;
      RETURN END.

   (sbpush) PROCEDURE(str1, str2, hed);
      % add pair to substitute list %
      LOCAL astr, len, subrp, asa, cap;
      POINTER hed, subrp, cap;
      REF str1, str2, astr;
      &astr ← hed.sbas;
      len ← astr.L;
      asa ← &astr + 1; %origin of text for A-string%
      *astr* ← *astr*, *str1*, *str2*;
      subrp ← hed.sbrp;
      subrp.carnc ← str1.L;
      subrp.catnc ← str2.L;
```

```
      subrp.carbp ← conbp(asa,len);
      subrp.catbp ← conbp(asa,len+str1.L);
      subrp.canxt ← 0;
      %link card into appropriate chain%
         cap ← hed.sbdp + *str2*[1];
         IF [cap] = 0 THEN %empty chain%
            [cap] ← subrp
         ELSE BEGIN %link on end%
            cap ← [cap];
            WHILE cap.canxt DO cap ← cap.canxt;
            cap.canxt ← subrp;
         END;
      hed.sbrp ← subrp + lcard;
      RETURN END.

   DECLARE bpary = (010677777777B, 3507B8, 2607B8, 1707B8, 1007B8);
   (conbp) PROCEDURE(base,cn);
      % construct byte pointer %
      LOCAL q, r;
      DIV cn / 5, q, r;
      RETURN(base+bpary[r]+q);
      END.

%.....Core-NLS Substitute Code.....%

   (subbranch) PROCEDURE (stid, sb, da); %substitute branch%
      REF sb, da;
      xsubgrp (stid, stid, &sb, &da);
      RETURN;
      END.

   (subplex) PROCEDURE (stid, sb, da); %substitute plex%
      LOCAL stid1, stid2;
      REF sb, da;
      stid1 ← plxset (stid : stid2);
      xsubgrp (stid1, stid2, &sb, &da);
      RETURN;
      END.

   (subgroup) PROCEDURE (stid1, stid2, sb, da); %substitute group%
      REF sb, da;
      stid1 ← grptst (stid1, stid2 : stid2);
      xsubgrp (stid1, stid2, &sb, &da);
      RETURN;
      END.

   (xsubgrp) PROCEDURE    % runs substitution over a group %
      (stid1,    % stid of first branch in group %
      stid2,     % stid of last branch in group %
      sb,        % address of patterns %
      da);       % display area %
      LOCAL stid,
         sw;    % address of sequence work area %
      REF da, sb, sw;
      IF &da NOT IN [$dpyarea, $dpyarea + da1*dacnt) OR
         NOT da.daexis OR da.daempty THEN err(0);
```

```
   dismes(1, $"Substitute In Progress");
   &sw ← openseq (stid1, stid2, da.davspec, da.davspc2, da.dausqcod,
   da.dacacode);
   WHILE (stid ← seqgen (&sw)) # endfil DO substat(stid, &sb, &da);
   closeseq (&sw);
   dismes (0);
   RETURN;
   END.

(substat) PROCEDURE(stid, sb, da);
   % substitute in one statement %
   LOCAL
      cary,   %pointer (later SKIPN) to subdsp%
      sfc,    %first char position for scan%
      sbp,    %byte pointer to chars in statement%
      snc,    %counter for chars in statement%
      ch,     %last char read from statement%
      chbp,   %byte ptr to end of last change%
      chnc,   %char pos of last change from right end of original
      statement%
      chflag, %count of changes made%
      cap,    %ptr to substitution description record%
      cbp,    %byte ptr to candidate or replacement%
      cnc,    %length of candidate or replacement%
      cncl,   %length of candidate%
      tbp,    %temp sbp for comparison%
      wbp,    %byte ptr to last char written in new statement%
      wnc,    %number of chars written in new statement%
      maxsl;  %max size of new statement%
   POINTER sb, da, cap;
   cary ← sb.sbdp;
   sfc ← IF stid.stastr OR da.davspec.vsnamf THEN 1 ELSE
   fchtxt(stid);
   sbp ← stbpget(stid, sfc: snc);
   IF sfc = 1 THEN BEGIN %entire statement%
      chbp ← sbp; chnc ← snc;
   END ELSE %name not scanned%
      chbp ← stbpget(stid, 1: chnc);
   chflag ← 0;
   maxsl ← lit.M;
   wbp ← 0107B8 + $lit; %text begins at $lit+1%
   wnc ← 0;
   % set up for fast scan %
      al ← skipni;
      !HLLM al,cary; %insert instruction part%
   (step1):
      BUMP snc;
   (step): %fast scan loop%
      !SOSG snc;
      !JRST done;
      !ILDB al,sbp;
      !XCT cary; %SKIPN a2,subdsp(al)%
      !JRST step;
      ch ← al; cap ← a2;
      DO BEGIN
         IF (cnc ← (cncl ← cap.catnc)-1) < snc THEN BEGIN
```

```
            tbp ← sbp;
            cbp ← cap.catbp;
            ch ← ↑cbp; %skip first char%
            FOR cnc DOWN UNTIL <=0 DO
               IF ↑tbp # ↑cbp THEN GOTO noteq;
            % found match %
            IF (wnc ← wnc + chnc - snc + (cnc ← cap.carnc))
               > maxsl THEN BEGIN
                  % generate string system overflow error %
                  lit.L ← lit.M;
                  *lit* ← *lit*, SP;
                  RETURN; %if no trap - may not be possible%
               END;
            % copy segment between last match and this %
            sbp ← sbp + (IF sbp < 3507B8 THEN 07B10 ELSE
            437777777777B);
               %back up sbp by 1 char%
            UNTIL chbp = sbp DO
               ↑wbp ← ↑chbp;
            % enter replacement %
            cbp ← cap.carbp;
            FOR cnc DOWN UNTIL <=0 DO
               ↑wbp ← ↑cbp;
            % update variables %
            chbp ← sbp ← tbp;
            chnc ← snc ← snc - cncl;
            BUMP chflag;
            GOTO stepl;
         END;
         (noteq):
      END WHILE cap ← cap.canxt;
      GOTO step;
   (skipni): !SKIPN a2,0(a1); %instruction part for cary%
   (done):
   IF chflag THEN BEGIN
      IF (wnc ← wnc + chnc) > maxsl THEN abort();
      % copy rest of statement (past last match) %
      UNTIL chbp = sbp DO
         ↑wbp ← ↑chbp;
      lit.L ← wnc;
      dsttx(stid); %remove markers%
      ST stid ← *lit*;
      subcnt ← subcnt + chflag;
   END;
   RETURN END.

(stbpget) PROCEDURE (stid, cnt); %statement byte pointer get%
   LOCAL bp;
   swork ← stid; swork[1] ← cnt;
   fechcl(1, $swork);
   %swork[2] contains byte count+1, swork[4] has byte pointer%
   bp ← swork[4];
   IF bp .A 77B10 = 44B10 THEN
      bp ← bp - 430000000001B;
      %adjust char and word pos%
   RETURN(bp, swork[2]-1) END.
```

FINISH of subst

TCMNDS

```
<NLS>TCMNDS.NLS;25, 3-JAN-72 17:17 MSC ;
FILE tcmnds   %  L10   <REL-NLS>TCMNDS.REL  %

%.....Declarations.....%

   REF tda;
   REGISTER r1=1, r2=2, r3=3;

%.....Append........%

   (ta) PROCEDURE;        %APPEND STATEMENT COMMAND%
      echo($"Append ");
      ON SIGNAL
         = sighelp :
            BEGIN
            typeas($" (to) ADDR CA (from) ADDR CA LIT CA");
            RETURN;
            END;
         ELSE;
      tobl();
      LOOP
         BEGIN
         fromb2();
         *lit* ← NULL;
         txtlit($lit);
         mvcbl();
         xas($b1, $b2, $lit);
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN;
      END.

%.....Break........%

   (tb) PROCEDURE;
      echo($"Break ");
      echo($"Statement");
      ON SIGNAL
         = sighelp :
            BEGIN
            typeas($" (at) ADDR CA LEVADJ CA");
            RETURN;
            END;
         ELSE;
      LOOP
         BEGIN
         prompt($" at ");
         tbug($b1);
         levadj(b1, $ladj);
         CASE curchr OF
            = CA, =C. : NULL;
            ENDCASE error();
         *lit* ← NULL;
         b1 ← xbs($b1, $ladj, $lit);
         b1[1] ← 1;
```

```
        mvcbl();
        IF curchr # C. THEN EXIT;
        crlf();
        END;
    RETURN;
    END.

%.....Copy........%

(tcmtl) PROCEDURE(proc);
    REF proc;
    ON SIGNAL
        = sighelp :
            BEGIN
            typeas($" (to) ADDR CA (from) ADDR CA");
            RETURN;
            END;
        ELSE;
    LOOP
        BEGIN
        tobl();
        fromb2();
        mvcbl();
        proc($bl, $b2);
        IF curchr # C. THEN EXIT;
        crlf();
        END;
    RETURN END.

(tcc) PROCEDURE;
    echo($"Character");
    tcmtl($xcc);
    RETURN END.

(tcw) PROCEDURE;
    echo($"Word");
    tcmtl($xcw);
    RETURN END.

(tcn) PROCEDURE;
    echo($"Number");
    tcmtl($xcn);
    RETURN END.

(tcv) PROCEDURE;
    echo($"Visible");
    tcmtl($xcv);
    RETURN END.

(tci) PROCEDURE;
    echo($"Invisible");
    tcmtl($xci);
    RETURN END.

(tcl) PROCEDURE;
    echo($"Link");
```

```
   tcmt1($xc1);
   RETURN END.

(tcmt2) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      = sighelp :
         BEGIN
         typeas($" (to) ADDR CA (from) ADDR CA ADDR CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      tobl();
      fromb2();
      prompt($" ");
      tbug($b3);
      mvcbl();
      proc($bl, $b2, $b3);
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tct) PROCEDURE;
   echo($"Text");
   tcmt2($xct);
   RETURN END.

(tcmsl) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      = sighelp :
         BEGIN
         typeas($" (to) ADDR CA (from) ADDR CA LEVADJ CA");
         RETURN;
         END;
      ELSE;
   tobl();
   LOOP
      BEGIN
      fromb2();
      levadj(bl, $ladj);
      CASE curchr OF
         = CA, =C. : NULL;
         ENDCASE error();
      bl ← proc($bl, $b2, $ladj);
      bl[1] ← 1;
      mvcbl();
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tcs) PROCEDURE;
```

```
   echo($"Statement");
   tcmsl($xcs);
   RETURN END.

(tcb) PROCEDURE;
   echo($"Branch");
   tcmsl($xcb);
   RETURN END.

(tcp) PROCEDURE;
   echo($"Plex");
   tcmsl($xcp);
   RETURN END.

(tcms2) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      = sighelp :
         BEGIN
         typeas($" (to) ADDR CA (from) ADDR CA ADDR CA LEVADJ CA");
         RETURN;
         END;
      ELSE;
   tobl();
   LOOP
      BEGIN
      fromb2();
      prompt($" ");
      tbug($b3);
      levadj(bl, $ladj);
      CASE curchr OF
         = CA, =C. : NULL;
         ENDCASE error();
      bl ← proc($bl, $b2, $b3, $ladj);
      bl[1] ← 1;
      mvcbl();
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tcg) PROCEDURE;
   echo($"Group");
   tcms2($xcg);
   RETURN END.

(tobl) PROCEDURE;
   prompt($" to ");
   tbug($bl);
   RETURN END.

(fromb2) PROCEDURE;
   prompt($" from ");
   tbug($b2);
   RETURN END.
```

```
%.....Delete........%

   (tdtl) PROCEDURE(proc);
      REF proc;
      ON SIGNAL
         = sighelp :
            BEGIN
            type$s($" (at) ADDR CA CA");
            RETURN;
            END;
         ELSE;
      LOOP
         BEGIN
         prompt($" at ");
         tbug($bl);
         prompt($" ok?");
         getctc();
         mvcbl();
         proc($bl);
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN END.

   (tdc) PROCEDURE;
      echo($"Character");
      tdtl($xdc);
      RETURN END.

   (tdw) PROCEDURE;
      echo($"Word");
      tdtl($xdw);
      RETURN END.

   (tdn) PROCEDURE;
      echo($"Number");
      tdtl($xdn);
      RETURN END.

   (tdv) PROCEDURE;
      echo($"Visible");
      tdtl($xdv);
      RETURN END.

   (tdi) PROCEDURE;
      echo($"Invisible");
      tdtl($xdi);
      RETURN END.

   (tdl) PROCEDURE;
      echo($"Link");
      tdtl($xdl);
      RETURN END.

   (tdt2) PROCEDURE(proc);
      REF proc;
```

```
    ON SIGNAL
       = sighelp :
          BEGIN
          typeas($" (at) ADDR CA ADDR CA CA");
          RETURN;
          END;
       ELSE;
    LOOP
       BEGIN
       prompt($" at ");
       tbug($b1);
       prompt($" ");
       tbug($b2);
       prompt($" ok?");
       getctc();
       mvcbl();
       proc($b1, $b2);
       IF curchr # C. THEN EXIT;
       crlf();
       END;
    RETURN END.

(tdt) PROCEDURE;
    echo($"Text");
    tdt2($xdt);
    RETURN END.

(tds) PROCEDURE;
    echo($"Statement");
    tdt1($xds);
    RETURN END.

(tdb) PROCEDURE;
    echo($"Branch");
    tdt1($xdb);
    RETURN END.

(tdp) PROCEDURE;
    echo($"Plex");
    tdt1($xdp);
    RETURN END.

(tdg) PROCEDURE;
    echo($"Group");
    tdt2($xdg);
    RETURN END.

%.....Fix........%

(tf) PROCEDURE;      %Fix Marker%
    ON SIGNAL
       = sighelp :
          BEGIN
          typeas($" LIT CA (at) ADDR CA");
          RETURN;
          END;
```

```
        ELSE;
      echo($"Fix marker");
      tfname();
      prompt($" at ");
      tbug($bl);
      mvcbl();
      xemf($bl, $lit);
      RETURN END.

   (tfname) PROCEDURE;
      prompt($" named ");
      echon();
      *lit* ← NULL;
      CASE lookc() OF % see if the first character is a ? %
         = SP:
            BEGIN
            input();
            REPEAT CASE;
            END;
         = '?:
            BEGIN
            input();
            SIGNAL(sighelp);
            END;
         ENDCASE;
      rdlit($lit, $rnmdel);
      RETURN END.

%.....Insert........%

   (titxt) PROCEDURE(proc);
      REF proc;
      ON SIGNAL
         = sighelp :
            BEGIN
            typeas($" (at) ADDR CA LIT CA");
            RETURN;
            END;
         ELSE;
      LOOP
         BEGIN
         prompt($" at ");
         tbug($bl);
         crlf();
         *lit* ← NULL;
         txtlit($lit);
         mvcbl();
         proc($bl, $lit);
         IF curchr # 0. THEN EXIT;
         crlf();
         END;
      RETURN END.

   (tic) PROCEDURE;
      echo($"Character");
      titxt($xic);
```

```
   RETURN END.

(tiw) PROCEDURE;
   echo($"Word");
   titxt($xiw);
   RETURN END.

(tin) PROCEDURE;
   echo($"Number");
   titxt($xin);
   RETURN END.

(tiv) PROCEDURE;
   echo($"Visible");
   titxt($xiv);
   RETURN END.

(tii) PROCEDURE;
   echo($"Invisible");
   titxt($xii);
   RETURN END.

(til) PROCEDURE;
   echo($"Link");
   titxt($xil);
   RETURN END.

(tit) PROCEDURE;
   echo($"Text");
   titxt($xic);
   RETURN END.

(tis) PROCEDURE;
   echo($"Statement");
   tinstructure();
   RETURN END.

(tinstructure) PROCEDURE;
   prompt($" at ");
   ON SIGNAL
      = sighelp :
         BEGIN
         typeas($" (at) ADDR CA LEVADJ LIT CA");
         RETURN;
         END;
      ELSE;
   tbug($bl);
   LOOP
      BEGIN
      levadj(bl, $ladj);
      *lit* ← NULL;
      txtlit($lit);
      bl ← xis($bl, $lit, $ladj);
      bl[1] ← 1;
      mvcbl();
      IF curchr # C. THEN RETURN;
```

```
        crlf();
        END;
    END.

%.....Move........%

    (tmc) PROCEDURE;
        echo($"Character");
        tcmt1($xmc);
        RETURN END.

    (tmw) PROCEDURE;
        echo($"Word");
        tcmt1($xmw);
        RETURN END.

    (tmn) PROCEDURE;
        echo($"Number");
        tcmt1($xmn);
        RETURN END.

    (tmv) PROCEDURE;
        echo($"Visible");
        tcmt1($xmv);
        RETURN END.

    (tmi) PROCEDURE;
        echo($"Invisible");
        tcmt1($xmi);
        RETURN END.

    (tml) PROCEDURE;
        echo($"Link");
        tcmt1($xml);
        RETURN END.

    (tmt) PROCEDURE;
        echo($"Text");
        tcmt2($xmt);
        RETURN END.

    (tms) PROCEDURE;
        echo($"Statement");
        tcms1($xms);
        RETURN END.

    (tmb) PROCEDURE;
        echo($"Branch");
        tcms1($xmb);
        RETURN END.

    (tmp) PROCEDURE;
        echo($"Plex");
        tcms1($xmp);
        RETURN END.
```

```
    (tmg) PROCEDURE;
       echo($"Group");
       tcms2($xmg);
       RETURN END.

%.....Name delimiters........%

    (tns1) PROCEDURE(proc);
       LOCAL STRING str1[4], strr[4];
       LOCAL dlleft, dlrght;
       REF proc;
       ON SIGNAL
          = sighelp :
             BEGIN
             typeas($" (at) ADDR CA");
             RETURN;
             END;
          ELSE;
       LOOP
          BEGIN
          prompt($" at ");
          tbug($bl);
          crlf();
          prompt($"Left delimiter ");
          *str1* ← NULL;
          txtlit($str1);
          crlf();
          prompt($"Right delimiter ");
          *strr* ← NULL;
          txtlit($strr);
          prompt($" ok? ");
          getctc();
          todco(CA);
          dlleft ←
             IF str1.L =empty OR (FIND SF(*str1*) ("null"/"NULL")) THEN 0
             ELSE *str1*[1];
          dlrght ←
             IF strr.L =empty OR (FIND SF(*strr*) ("NULL"/"null")) THEN 0
             ELSE *strr*[1];
          proc($bl, dlleft, dlrght, tda);
          IF curchr # C. THEN EXIT;
          crlf();
          END;
       RETURN END.

    (tns2) PROCEDURE(proc);
       LOCAL STRING str1[4], strr[4];
       LOCAL dlleft, dlrght;
       REF proc;
       ON SIGNAL
          = sighelp :
             BEGIN
             typeas($" (at) ADDR CA");
             RETURN;
             END;
          ELSE;
```

```
   LOOP
      BEGIN
      prompt($" at ");
      tbug($b1);
      prompt($" ");
      tbug($b2);
      crlf();
      prompt($"Left delimiter ");
      *str1* ← NULL;
      txtlit ($str1);
      crlf();
      prompt($"Right delimiter ");
      *strr* ← NULL;
      txtlit ($strr);
      prompt($" ok? ");
      getctc();
      todco(CA);
      dlleft ←
         IF str1.L =empty OR (FIND SF(*str1*) ("null"/"NULL")) THEN 0
         ELSE *str1*[1];
      dlrght ←
         IF strr.L =empty OR (FIND SF(*strr*) ("NULL"/"null")) THEN 0
         ELSE *strr*[1];
      mvcbl();
      proc($b1, dlleft, dlrght, tda);
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tnd) PROCEDURE;
   LOCAL STRING str[15];
   echo($"Display");
   ON SIGNAL
      =sighelp:
         BEGIN
         typeas($" (at) ADDR CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      prompt($" at ");
      tbug($b1);
      mvcbl();
      <AUXCOD, getnmdls>(b1, $str);
      crlf();
      typeas($str);
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tns) PROCEDURE;
   echo($"Statement");
   tnsl($xns);
```

```
    RETURN END.

(tnb) PROCEDURE;
    echo($"Branch");
    tnsl($xnb);
    RETURN END.

(tnp) PROCEDURE;
    echo($"Plex");
    tnsl($xnp);
    RETURN END.

(tng) PROCEDURE;
    echo($"Group");
    tns2($xng);
    RETURN END.

%.....Print........%

(tpr) PROCEDURE(type);
     ON SIGNAL
       =sighelp:
          BEGIN
          typeas($" (at) ADDR CA VIEWSPECS CA");
          RETURN;
          END;
       ELSE;
    tbug($bl);
    getvsp();
    treslev(bl);
    <TSPRT, printg>(bl, bl, type);
    RETURN;
    END.

(tps) PROCEDURE;
    echo($"Statement ");
    tpr(stmtv);
    RETURN;
    END.

(tpb) PROCEDURE;
    echo($"Branch ");
    tpr(brnchv);
    RETURN;
    END.

(tpp) PROCEDURE;
    LOCAL stid1, stid2;
    echo($"Plex ");
    ON SIGNAL
       =sighelp:
          BEGIN
          typeas($" (at) ADDR CA VIEWSPECS CA");
          RETURN;
          END;
       ELSE;
```

```
      tbug($b1);
      getvsp();
      treslev(b1);
      stid1 ← plxset(b1 : stid2);
      <TSPT, printg>(stid1, stid2, plexv);
      RETURN;
      END.

   (tpg) PROCEDURE;
      ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA ADDR CA VIEWSPECS CA");
            RETURN;
            END;
         ELSE;
      echo($"Group ");
      tbug($b1);
      prompt($" ");
      tbug($b2);
      b1 ← <STRMNP, grptst>(b1, b2 : b2);
      getvsp();
      treslev(b1);
      <TSPRT, printg>(b1, b2, groupv);
      RETURN;
      END.

%.....Replace........%

   (trt1) PROCEDURE(proc);
      REF proc;
      ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA (by literal?) Yes LIT or No ADDR
            CA");
            GOTO STATE;
            END;
         ELSE;
      LOOP
         BEGIN
         treptext();
         mvcbl();
         proc(rplb, $b1, $b2, $lit);
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN END.

   (treptext) PROCEDURE;
      LOCAL char;
      prompt($" at ");
      tbug($b1);
      IF treptype() THEN
         BEGIN
         crlf();
```

```
      *lit* ← NULL;
      txtlit($lit);
      rplb ← FALSE;
      END
   ELSE
      BEGIN
      tbug($b2);
      rplb ← TRUE;
      END;
   RETURN END.

(trc) PROCEDURE;
   echo($"Character");
   trtl($xrc);
   RETURN END.

(trw) PROCEDURE;
   echo($"Word");
   trtl($xrw);
   RETURN END.

(trn) PROCEDURE;
   echo($"Number");
   trtl($xrn);
   RETURN END.

(trv) PROCEDURE;
   echo($"Visible");
   trtl($xrv);
   RETURN END.

(tri) PROCEDURE;
   echo($"Invisible");
   trtl($xri);
   RETURN END.

(trl) PROCEDURE;
   echo($"Link");
   trtl($xrl);
   RETURN END.

(trt2) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      =sighelp:
         BEGIN
         *lit* ← EOL,
            " (at) ADDR CA ADDR CA", EOL,
            " (by literal?) Yes LIT or No ADDR CA ADDR CA";
         typeas($lit);
         GOTO STATE;
         END;
      ELSE;
   LOOP
      BEGIN
      prompt($" at ");
```

```
      tbug($b1);
      prompt($" ");
      tbug($b2);
      IF treptype() THEN
         BEGIN
         crlf();
         *lit* ← NULL;
         txtlit($lit);
         rplb ← FALSE;
         END
      ELSE
         BEGIN
         tbug($b3);
         prompt($" ");
         tbug($b4);
         rplb ← TRUE;
         END;
      mvcbl();
      proc(rplb, $b1, $b2, $b3, $b4, $lit);
      IF curchr # C. THEN EXIT;
      END;
   RETURN END.

(trt) PROCEDURE;
   echo($"Text");
   trt2($xrt);
   RETURN END.

(treptype) PROCEDURE;
   prompt($" by literal? ");
   echoff();
   CASE inpcuc() OF
      =SP, =CA, ='Y:
         BEGIN
         prompt($" Yes");
         RETURN(TRUE);
         END;
      ='N:
         BEGIN
         prompt($" No");
         RETURN(FALSE);
         END;
      =CD: GOTO STATE;
      ENDCASE error();
   END.
(trsl) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      =signelp:
         BEGIN
         typeas($" (at) ADDR CA (by literal?) Yes LIT or No ADDR
         CA");
         GOTO STATE;
         END;
      ELSE;
   treptext();
```

```
   bl ← proc(rplb, $bl, $b2, $lit);
   bl[1] ← 1;
   mvcbl();
   trepstructure();
   RETURN END.

(trs) PROCEDURE;
   echo($"Statement");
   trsl($xrs);
   RETURN END.

(trb) PROCEDURE;
   echo($"Branch");
   trsl($xrb);
   RETURN END.

(trp) PROCEDURE;
   echo($"Plex");
   trsl($xrp);
   RETURN END.

(trg) PROCEDURE;
   LOCAL char;
   ON SIGNAL
      =sighelp:
         BEGIN
         *lit* ← EOL,
            " (at) ADDR CA ADDR CA", EOL,
            " (by text?) Yes TEXT or No ADDR CA ADDR CA";
         typeas($lit);
         GOTO STATE;
         END;
      ELSE;
   echo($"Group");
   prompt($" at ");
   tbug($bl);
   prompt($" ");
   tbug($b2);
   IF treptype() THEN
      BEGIN
      crlf();
      *lit* ← NULL;
      txtlit($lit);
      rplb ← FALSE;
      END
   ELSE
      BEGIN
      tbug($b3);
      prompt($" ");
      tbug($b4);
      rplb ← TRUE;
      END;
   bl ← xrg(rplb, $bl, $b2, $b3, $b4, $lit);
   bl[1] ← 1;
   mvcbl();
   trepstructure();
```

```
    RETURN END.

(trepstructure) PROCEDURE;
    WHILE curchr = C. DO
        BEGIN
        crlf();
        levadj(bl, $ladj);
        *lit* ← NULL;
        txtlit($lit);
        bl ← xis($bl, $lit, $ladj);
        bl[1] ← 1;
        mvcbl();
        END;
    RETURN END.

%.....Substitute........%

(tssl) PROCEDURE(type);
    ON SIGNAL
        =sighelp:
            BEGIN
            *lit* ← EOL,
                " (at) ADDR CA", EOL,
                " (literal) LIT CA (for literal) LIT CA (go?)", EOL,
                " Yes or No (literal) ... ";
            typeas($lit);
            GOTO STATE;
            END;
        ELSE;
    LOOP
        BEGIN
        prompt($" at ");
        tbug($bl);
        tsubcn(type, bl);
        bl[1] ← 1;
        mvcbl();
        IF curchr # C. THEN EXIT;
        crlf();
        END;
    RETURN END.

(tsubs) PROCEDURE;
    echo($"Statement");
    tssl(stmtv);
    RETURN END.

(tsubb) PROCEDURE;
    echo($"Branch");
    tssl(brnchv);
    RETURN END.

(tsubp) PROCEDURE;
    LOCAL stid1, stid2;
    ON SIGNAL
        =sighelp:
            BEGIN
```

```
            *lit* ← EOL,
               " (at) ADDR CA", EOL,
               " (literal) LIT CA (for literal) LIT CA (go?) Yes or No";
            typeas($lit);
            GOTO STATE;
            END;
         ELSE;
      echo($"Plex");
      LOOP
         BEGIN
         prompt($" at ");
         tbug($bl);
         stid1 ← <STRMNP, plxset> (bl : stid2);
         tsubcn(plexv, stid1, stid2);
         bl[1] ← 1;
         mvcbl();
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN END.

   (tsubg) PROCEDURE;
      LOCAL stid1, stid2;
      ON SIGNAL
         =sighelp:
            BEGIN
            *lit* ← EOL,
               " (at) ADDR CA ADDR CA", EOL,
               " (literal) LIT CA (for literal) LIT CA (go?) Yes or No";
            typeas($lit);
            GOTO STATE;
            END;
         ELSE;
      echo($"Group");
      LOOP
         BEGIN
         prompt($" at ");
         tbug($bl);
         prompt($" ");
         tbug($b2);
         stid1 ← <STRMNP, grptst> (bl, b2 : stid2);
         tsubcn(groupv, stid1, stid2);
         bl[1] ← 1;
         mvcbl();
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN END.

   (tsubcn) PROCEDURE(type, stid1, stid2);
      %-----------------%
      LOCAL count;
      LOCAL STRING rstr[80], tstr[80];
      LOCAL subdsp[200B];
      *auxlit* ← NULL;
      count ← 0;
```

```
      CALL sbinit($subhed, $subrec, $subdsp, $auxlit);
      LOOP  %collect patterns%
         BEGIN
         BUMP count;
         crlf();
         *rstr* ← NULL;
         prompt($" Literal ");
         txtlit($rstr); %get pattern%
         *tstr* ← NULL;
         prompt($" For ");
         txtlit($tstr);
         <SUBST, sbpush>($rstr, $tstr, $subhed); %push strings%
         IF count = $subnum THEN EXIT;
         prompt($" Go? ");
         IF answer() THEN EXIT;
         END;
      crlf();
      <SUBST, substitute>(type, stid1, stid2, $subhed, &tda);
      crlf();
      typeas($"substitutions = ");
      typeno(subcnt, 0); %type number of substitutions%
      RETURN;
      END.

%.....Transpose........%

   (ttt1) PROCEDURE(proc);
      REF proc;
      ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA (and) ADDR CA");
            RETURN;
            END;
         ELSE;
      LOOP
         BEGIN
         prompt($" at ");
         tbug($b1);
         prompt($" and ");
         tbug($b2);
         mvcbl();
         proc($b1, $b2);
         IF curchr # 0. THEN EXIT;
         crlf();
         END;
      RETURN END.
   (ttc) PROCEDURE;
      echo($"Character");
      ttt1($xtc);
      RETURN END.

   (ttw) PROCEDURE;
      echo($"Word");
      ttt1($xtw);
      RETURN END.
```

```
(ttn) PROCEDURE;
   echo($"Number");
   ttt1($xtn);
   RETURN END.

(ttv) PROCEDURE;
   echo($"Visible");
   ttt1($xtv);
   RETURN END.

(tti) PROCEDURE;
   echo($"Invisible");
   ttt1($xti);
   RETURN END.

(ttl) PROCEDURE;
   echo($"Link");
   ttt1($xtl);
   RETURN END.

(ttt2) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      =sighelp:
         BEGIN
         typeas($" (at) ADDR CA ADDR CA (and) ADDR CA ADDR CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      prompt($" at ");
      tbug($b1);
      prompt($" ");
      tbug($b2);
      prompt($" and ");
      tbug($b3);
      prompt($" ");
      tbug($b4);
      mvcbl();
      proc($b1, $b2, $b3, $b4);
      IF curchr # 0. THEN EXIT;
      crlf();
      END;
   RETURN END.
(ttt) PROCEDURE;
   echo($"Text");
   ttt2($xtt);
   RETURN END.

(tts) PROCEDURE;
   echo($"Statement");
   ttt1($xts);
   RETURN END.
```

```
    (ttb) PROCEDURE;
       echo($"Branch");
       ttt1($xtb);
       RETURN END.

    (ttp) PROCEDURE;
       echo($"Plex");
       ttt1($xtp);
       RETURN END.

    (ttg) PROCEDURE;
       echo($"Group");
       ttt2($xtg);
       RETURN END.

%.....view specs........%

    (tv) PROCEDURE;
       echo($"View specs: ");
       ON SIGNAL
          =sighelp:
             BEGIN
             *lit* ← EOL,
                " a   (L ← L-1)", EOL,
                " b   (L ← L+1)", EOL,
                " c   (L ← ALL)", EOL,
                " d   (L ← 1)", EOL,
                " e   (L ← REL)", EOL,
                " g   (branch only on)", EOL,
                " h   (branch only off)", EOL,
                " i   (content analyzer on)", EOL;
             *lit* ← *lit*,
                " j   (content analyzer off)", EOL,
                " k   (content analyzer fail)", EOL,
                " l   (plex only)", EOL,
                " m   (statement numbers on)", EOL,
                " n   (statement numbers off)", EOL,
                " q   (T ← T-1)", EOL,
                " r   (T ← T+1)", EOL,
                " s   (T ← ALL)", EOL,
                " w   (L ← T ← ALL)", EOL;
             *lit* ← *lit*,
                " x   (L ← T ← 1)", EOL,
                " y   (blank lines on)", EOL,
                " z   (blank lines off)", EOL,
                " A   (indenting on)", EOL,
                " B   (indenting off)", EOL,
                " C   (names on)", EOL,
                " D   (names off)", EOL,
                " K   (signiture on)", EOL,
                " L   (signiture off)";
             typeas($lit);
             RETURN;
             END;
          ELSE;
       getvsp();
```

```
      treslev(tda.dacsp); %resolve relative level wrt dacsp%
      RETURN;
      END.

%.....Xset........%

   (txtl) PROCEDURE(proc);
      REF proc;
      ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA");
            RETURN;
            END;
         ELSE;
      LOOP
         BEGIN
         prompt($" at ");
         tbug($bl);
         mvcbl();
         proc($bl);
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN END.
   (txc) PROCEDURE;
      echo($"Character");
      txtl($xxc);
      RETURN END.

   (txw) PROCEDURE;
      echo($"Word");
      txtl($xxw);
      RETURN END.

   (txn) PROCEDURE;
      echo($"Number");
      txtl($xxn);
      RETURN END.

   (txv) PROCEDURE;
      echo($"Visible");
      txtl($xxv);
      RETURN END.

   (txi) PROCEDURE;
      echo($"Invisible");
      txtl($xxi);
      RETURN END.

   (txl) PROCEDURE;
      echo($"Link");
      txtl($xxl);
      RETURN END.

   (txt2) PROCEDURE(proc);
```

```
   REF proc;
   ON SIGNAL
      =sighelp:
         BEGIN
         typeas($" (at) ADDR CA ADDR CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      prompt($" at ");
      tbug($b1);
      prompt($" ");
      tbug($b2);
      mvcbl();
      proc($b1, $b2);
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.
(txt) PROCEDURE;
   echo($"Text");
   txt2($xxt);
   RETURN END.

(txs) PROCEDURE;
   echo($"Statement");
   txt1($xxs);
   RETURN END.

(txb) PROCEDURE;
   echo($"Branch");
   txt1($xxb);
   RETURN END.

(txg) PROCEDURE;
   echo($"Group");
   txt2($xxg);
   RETURN END.

(txp) PROCEDURE;
   echo($"Plex");
   txt1($xxp);
   RETURN END.

(txm) PROCEDURE;
   CASE inpcuc() OF
      = 'C:
         BEGIN
         echo($"Capital ");
         xsucm();
         REPEAT;
         END;
      = 'L:
         BEGIN
         echo($"Lower ");
```

```
            xslcm();
            REPEAT;
            END;
         = '?:
            BEGIN
            *lit* ←
               EOL,
               "Capital ", EOL,
               "Lower ";
            typeas($lit);
            RETURN;
            END;
         = SP: REPEAT CASE;
         = CD: GOTO STATE;
         = CA:
            BEGIN
            todco(CA);
            RETURN;
            END;
         ENDCASE error();
      END.

FINISH
```

TCTL

```
<NLS>TCTL.NLS;94, 2-JAN-72 22:05 BLP ;
FILE tctl   %  L10   <REL-NLS>TCTL.REL  %

REF tda;

%.....Primary Command Language Routine.....%

   (mainct)PROCEDURE; %command language for todas%
      %-----------------%
      LOCAL count; %for feedback line indenting%
      ON SIGNAL =-4: %Called from rubout% reset(); ELSE SIGNAL;
      <INPFBK, croot>();
      LOOP
         BEGIN
         crlf();
         count ← 0;
         WHILE (count ← count + 1) <= fedind DO typech(SP);
         typech('*);
         echoff();
         input();
         ttywc();
         END;
      END.

   (ttywc)
      %This routine determines the command being specified and transfers
      control to the appropriate routine.%
      PROCEDURE;
      LOCAL char, newversion;
      IF (char ← curchr) IN ['a, 'z] THEN char ← char - 40B;
      CASE char OF
         = 'A:  %append%
            <TCMNDS, ta> ();
         = 'B:  %break%
            <TCMNDS, tb> ();
         = 'C:  %copy%
            BEGIN
            echo($"Copy ");
            CASE inpcuc() OF
               = 'C: tcc(); %copy character%
               = 'W: tcw(); %copy word%
               = 'N: tcn(); %copy number%
               = 'V: tcv(); %copy visible%
               = 'I: tci(); %copy invisible%
               = 'L: tcl(); %copy link%
               = 'T: tct(); %copy text%
               = 'S: tcs(); %copy statement%
               = 'B: tcb(); %copy branch%
               = 'P: tcp(); %copy plex%
               = 'G: tcg(); %copy group%
               = '?: tyhelp(0); %type entities for edit commands%
               = SP: REPEAT CASE;
               ENDCASE error();
            END;
         = 'D:              %Delete Command%
            BEGIN
```

```
        echo($"Delete ");
        CASE inpcuc() OF
            = 'C: tdc(); %delete character%
            = 'W: tdw(); %delete word%
            = 'N: tdn(); %delete number%
            = 'V: tdv(); %delete visible%
            = 'I: tdi(); %delete invisible%
            = 'L: tdl(); %delete link%
            = 'T: tdt(); %delete text%
            = 'S: tds(); %delete statement%
            = 'B: tdb(); %delete branch%
            = 'P: tdp(); %delete plex%
            = 'G: tdg(); %delete group%
            = '?: tyhelp(0); %type entities for edit commands%
            = SP: REPEAT CASE;
            ENDCASE error();
        END;
    = 'E: <TXCT, texecute>(); %Execute Command%
    = 'F: <TCMNDS, tf>();          %Fix Marker%
    = 'G: <TXCT, tgoto>();      %Goto comand%
    = 'I:                      %Insert Command%
        BEGIN
        echo($"Insert ");
        CASE inpcuc() OF
            = 'C: tic(); %insert character%
            = 'W: tiw(); %insert word%
            = 'N: tin(); %insert number%
            = 'V: tiv(); %insert visible%
            = 'I: tii(); %insert invisible%
            = 'L: til(); %insert link%
            = 'T: tit(); %insert text%
            = 'S,
            = 'B,
            = 'P,
            = 'G: tis(); %insert statement%
            = '?:
                BEGIN
                *lit* ← EOL, "Statement", EOL;
                tyhp(2);
                typeas($lit);
                END;
            = SP: REPEAT CASE;
            ENDCASE error();
        END;
    = 'L: <IOCTL, tl> ();              %Load Command%
    = 'M:                      %Move Command%
        BEGIN
        echo($"Move ");
        CASE inpcuc() OF
            = 'C: tmc(); %move character%
            = 'W: tmw(); %move word%
            = 'N: tmn(); %move number%
            = 'V: tmv(); %move visible%
            = 'I: tmi(); %move invisible%
            = 'L: tml(); %move link%
            = 'T: tmt(); %move text%
```

```
        = 'S: tms(); %move statement%
        = 'B: tmb(); %move branch%
        = 'P: tmp(); %move plex%
        = 'G: tmg(); %move group%
        = '?: tyhelp(0); %type entities for edit commands%
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= 'N:                   %Name delimiter Command%
    BEGIN
    echo($"Name delimiter ");
    CASE inpcuc() OF
        = 'D: tnd(); %Name delimiter display%
        = 'S: tns(); %Name delimiter statement%
        = 'B: tnb(); %Name delimiter branch%
        = 'P: tnp(); %Name delimiter plex%
        = 'G: tng(); %Name delimiter group%
        = '?:
            BEGIN
            *lit* ←
                EOL,
                "Display ";
            tyhp(1);
            typeas($lit);
            END;
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= 'O: <IOCTL, to> ();                   %Output Command%
= 'P:                           %Print Command%
    BEGIN
    echo($"Print ");
    CASE inpcuc() OF
        = 'S: tps(); %print statement%
        = 'B: tpb(); %print branch%
        = 'P: tpp(); %print plex%
        = 'G: tpg(); %print group%
        = '?: tyhelp(1); %type structural entities%
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= 'R:                           %Replace Command%
    BEGIN
    echo($"Replace ");
    CASE inpcuc() OF
        = 'C: trc(); %replace character%
        = 'W: trw(); %replace word%
        = 'N: trn(); %replace number%
        = 'V: trv(); %replace visible%
        = 'I: tri(); %replace invisible%
        = 'L: trl(); %replace link%
        = 'T: trt(); %replace text%
        = 'S: trs(); %replace statement%
        = 'B: trb(); %replace branch%
        = 'P: trp(); %replace plex%
        = 'G: trg(); %replace group%
```

```
        = '?: tyhelp(0); %type entities for edit commands%
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= 'S: %<TSPRT,sbstut>(); Substitute Command%
    BEGIN
    echo($"Substitute ");
    CASE inpcuc() OF
        = 'S: tsubs(); %substitute statement%
        = 'B: tsubb(); %substitute branch%
        = 'P: tsubp(); %substitute plex%
        = 'G: tsubg(); %substitute group%
        = '?: tyhelp(1); %type structural entities%
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= 'T:                   %Transpose Command%
    BEGIN
    echo($"Transpose ");
    CASE inpcuc() OF
        = 'C: ttc(); %transpose character%
        = 'W: ttw(); %transpose word%
        = 'N: ttn(); %transpose number%
        = 'V: ttv(); %transpose visible%
        = 'I: tti(); %transpose invisible%
        = 'L: ttl(); %transpose link%
        = 'T: ttt(); %transpose text%
        = 'S: tts(); %transpose statement%
        = 'B: ttb(); %transpose branch%
        = 'P: ttp(); %transpose plex%
        = 'G: ttg(); %transpose group%
        = '?: tyhelp(0); %type entities for edit commands%
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= 'U: <IOCTL, tuf>();    %Update File%
= 'V: <TCMNDS,tv>();     %View specs Command%
= 'X:                    %Xset Command%
    BEGIN
    echo($"Xset ");
    CASE inpcuc() OF
        = 'C: txc(); %set character%
        = 'W: txw(); %set word%
        = 'N: txn(); %set number%
        = 'V: txv(); %set visible%
        = 'I: txi(); %set invisible%
        = 'L: txl(); %set link%
        = 'T: txt(); %set text%
        = 'S: txs(); %set statement%
        = 'B: txb(); %set branch%
        = 'P: txp(); %set plex%
        = 'G: txg(); %set group%
        = 'M: %set mode%
            BEGIN
            echo($"Mode ");
            txm();
```

```
                END;
            = '?:
                BEGIN
                *lit* ←
                    EOL,
                    "Mode ";
                typeas($lit);
                tyhelp(0);
                END;
            = SP: REPEAT CASE;
            ENDCASE error();
        END;
    = '.: tpoint();             %Show point%
    = ';: tcmnt();              %Comment Command%
    = '/:                       %Slash Command%
        BEGIN
        todco('/);
        tslash(tda.dacsp, tda.daccnt);
        END;
    = '\: tbslash();            %Backslash Command%
    = '↑: tua();                %Up Arrow Command%
    = LF: tlinfed();            %Line Feed Command%
    = SP: taddr();              %Get new address%
    = $ascalt: taltm();         %Repeat Search%
    =CD, =CA, =EOL, =CR, =C.: NULL;
    =todlit:
        REPEAT CASE (char ← rawchr());
    = '?: tyhelp(-1);           %Help%
    ENDCASE error();
RETURN;
END.
```

FINISH of tctl

TSPRT

```
<NLS>TSPRT.NLS;30, 14-MAR-72 22:33 BLP ;

FILE tsprt  %  L10  <REL-NLS>TSPRT  %

%.....Declarations.................%

   REGISTER r1 = 1;
   REF tda;
   DECLARE STRING dashes = "----------";

   %.....Global Declarations.........%

      % GLOBAL typnwa, typend, feedbk, fedind, caflg, sublnk, buffsz,
      buffct, buff, curchr, todlit, trnsli, trnlso, pshift, tshift,
      cshft0, wshft0, pshft0, tablst, subcnt, textex, textfg, rdtext,
      stn2, swork, tprtpt;%

%.....Print Routines..............%

   (printg) PROCEDURE (stid1, stid2, type);
      %print the structure (type) addrssed by stid1,stid2%
      %------------------%
      LOCAL
         char,   %current character%
         gapcol, %column of character preceding last invisible char%
         gapptr, %pointer to character preceding last invisible char%
         startp, %pointer to start of line character%
         prindt,  %number of columns to indent%
         lincnt, %count of lines printed for current statement%
         stid,   %current stid being printed%
         maxcol, %maximum number of columns for current line%
         stnrt,  %whether statement numbers are to be printed on right%
         sw,     %address of sequence generator work area%
         head,   %address of file header%
         mkrptr; %marker table pointer%
         LOCAL STRING
            str[100],    %scratch string%
            stnsig[50];  %holds statement number or signature%
         REF sw, mkrptr;

      IF stid1 = endfil THEN error();
      lineno ← IF type = stmtv THEN 0    %do not eject page%
         ELSE tda.damrow / tda.davinc;   %set to eject page%
      &sw ←
         <SEQGEN, openseq> (stid1, stid2, tda.davspec, tda.davspc2,
         tda.dausqcod, tda.dacacode);
      b1[0] ← IF stid1.stastr THEN tda.dacsp ELSE stid1;
      UNTIL ((stid ← seqgen (&sw)) = endfil) OR
        (type = stmtv AND sw.swcstid # stid1) DO
         BEGIN
         %set up work area s2work for READC%
            s2work ← stid;
            s2work[1] ←
               IF tda.davspec.vsnamf OR stid.stastr THEN 1 %print name%
               ELSE fchtxt (getsdb (stid)); %skip past name%
            fechcl (forward, $s2work);
```

```
%markers (not allowed for now)%
   prmkrf ← 0;
   IF FALSE AND tda.davspec.vsmkrf AND NOT stid.stastr THEN
      BEGIN  %see if marker in statemnt%
      &mkrptr ← $mkrtb - $filhed +
         (head ← <FILMNP, filhdr>(stid.stfile));
      mkrend ← &mkrptr +
         (($mkrtbl - $filhed + head) * mkrl;
      mkrflg ← FALSE;
      FOR mkrptr UP mkrl UNTIL = mkrend DO
         IF mkrptr.mkpsid = stid.stpsid THEN
            BEGIN
            mkrflg ← TRUE;
            mkrcnt ← mkrptr.mkccnt + 1;
            EXIT;
            END;
      END;
prindt ←        %indentation%
   IF tda.davspec.vsindf THEN
      MAX (0, MIN (tda.daind * (sw.swclvl-1), tda.damind,
      spst.M))
   ELSE 0;
*prbuf* ← *spst* (empty + 1 TO prindt);
maxcol  ← tda.damcol;   %max no. cols%
stnrt   ← FALSE;
IF tda.davspec.vsstnf AND stid.stpsid # origin AND
 NOT stid.stastr THEN
   IF NOT tda.davspec.vsstnr THEN
      BEGIN  %print statement number%
      fechnm (sw.swsvw, $prbuf);
      *prbuf* ← *prbuf*, SP;
      END
   ELSE   % statement numbers go on the right %
      BEGIN
      *stnsig* ← NULL;
      fechnm (sw.swsvw, $stnsig);
      maxcol ← maxcol - stnsig.L - 1;
      stnrt  ← TRUE;
      END;
%line printing loop%
FOR lincnt ← 0 UP UNTIL = tda.davspec.vstrnc DO
   BEGIN
   IF inptrf THEN EXIT 2;
   gapcol ← tda.daccol ← prbuf.L * tda.dahinc;
   gapptr ← startp ← prbuf.L;
   UNTIL tda.daccol >= maxcol DO
      CASE char ← READC ($s2work) OF
         =SP:
            BEGIN
            gapptr     ← prbuf.L;
            gapcol     ← tda.daccol;
            *prbuf*    ← *prbuf*, SP;
            tda.daccol ← tda.daccol + 1;
            END;
         =ENDCHR, =EOL, =CR:
            BEGIN
```

tda.daccol ← tda.daccol + putchr(char, tda.daccol, &prbuf);

```
                gapptr      ← prbuf.L;
                gapcol      ← tda.daccol;
                EXIT LOOP;
                END;
             =TAB:
                BEGIN
                gapptr      ← prbuf.L;
                gapcol      ← tda.daccol;
                (*prbuf*    ← *prbuf*, TAB:
                tda.daccol ← fndtab (&tda, tda.daccol + 1);
                END;
             ENDCASE
                BEGIN
                (*prbuf* ← *prbuf*, char;
                tda.daccol ← tda.daccol + 1;)
                END;
          %by here, a line has been constructed. fix gapptr and
          gapcol if there were no invisibles%
             IF startp = gapptr THEN   %no in invisibles in line%
                BEGIN
                gapptr ← prbuf.L;
                gapcol ← tda.daccol;
                END;
          IF stnrt THEN
             BEGIN
             stnrt    ← FALSE;
             maxcol   ← tda.damcol;   %max no. cols%
             *str*     ← *prbuf* [gapptr + 2 TO prbuf.L];
             *prbuf* [gapptr + 1 TO prbuf.L] ←
                *spst* [gapcol + 1 TO maxcol - stnsig.L],
                *stnsig*;
             IF NOT prtype (prbuf.L) THEN EXIT 2; %type the line%
             *prbuf* [prindt + 1 TO prbuf.L] ← *str*;
             END
          ELSE
             BEGIN
             IF NOT prtype (gapptr) THEN EXIT 2; %type the line%
             *prbuf* [prindt + 1 TO prbuf.L] ←
                *prbuf* [gapptr + 2 TO prbuf.L];
             END;
          bl[0] ← stid;
          IF char = ENDCHR THEN EXIT;
          END;
       %blank line and signature%
          IF tda.davspec.vsidtf AND NOT stid.stastr THEN
             BEGIN
             *stnsig* ← NULL;
             fechsig (stid, $stnsig);
             *prbuf* ←
                *spst* [1 TO maxcol - stnsig.L],
                *stnsig*;
             prtype (prbuf.L);
             END
          ELSE IF tda.davspec.vsblkf THEN crlf();
       END;
    <SEQGEN, closeseq> (&sw);
```

```
      crlf ();
      bl[1] ← 1;
      mvcbl(); %Move control marker to bl%
      RETURN;
      END.

   (prtype) PROCEDURE (number);
      %type 'number' characters from prbuf%
      %-----------------%
      LOCAL length; %length of astring in prbuff%
      %wait for output buffer empty%
         rl ← 101B; !JSYS dobe;
      IF inptrf %↑0% THEN RETURN (FALSE);
      IF inpstp %↑S% THEN
         BEGIN
         inpstp ← FALSE;
         RETURN (FALSE);
         END;
      IF tda.davspec.vspagf AND tda.dacrow >= tda.damrow THEN newpag()
      ELSE crlf();
      tda.dacrow ← tda.dacrow + tda.davinc;
      length ← prbuf.L := number;
      typeas ($prbuf);
      prbuf.L ← length;
      RETURN (TRUE);
      END.

   (newpag)  %page eject%
      % this procedure does a page eject for printing in todas. It
      takes into account the device being used, and tries to make the
      total page length equal to 11 1/2 inches, unless otherwise
      specified by the user%
      %-----------------%
      PROCEDURE;
      LOCAL top, bottom, extra;
      DIV (tda.dabottom-tda.datop-tda.damrow)/2, top, extra;
      bottom ← top + extra;
         %put extra line (if it exists) at the bottom%
      crlf();
      DO crlf() UNTIL (bottom ← bottom - tda.davinc) <=
      tda.davinc;
      IF bottom > 0 THEN
         BEGIN
         typeas($dashes);
         crlf();
         END;
      UNTIL (top := top - tda.davinc) <= 0 DO crlf();
      tda.dacrow ← 0;
      RETURN;
      END.

%.....Verify command code.........%

   (todvfy) PROCEDURE; %todas entry to File Verify%
      %-----------------%
      LOCAL vrfy;
```

```
    LOOP
       BEGIN
       echo($"Verify ");
       IF answer() THEN
          BEGIN
          vrfy ← TRUE;
          EXIT;
          END;
       crlf();
       prompt($"Checksum ");
       IF answer() THEN
          BEGIN
          vrfy ← FALSE;
          EXIT;
          END;
       crlf()
       END;
    crlf();
    <VERIFY, vfmain>(tda.dacsp.stfile, vrfy);
    GOTO STATE;
    END.

%.....secondary i/o routines.....%

   (answer) %this procedure accepts a yes or no answer from the
   keyboard, and returns with a 0 if the answer was negative, and a 1
   if it was positive%
      %------------------%
      PROCEDURE;
      echoff();
      CASE inpcuc() OF
         = 'Y, = CA:
            BEGIN
            echo($"Yes");
            RETURN(TRUE);
            END;
         = 'N, = SP:
            BEGIN
            echo($"No");
            RETURN(FALSE)
            END;
         = CD: GOTO STATE;
         ENDCASE
            BEGIN
            typeas($" ?");
            REPEAT;
            END;
      END.

   (gttnum) PROCEDURE (oldvlu);
      %read a number for TODAS from tty, accepting the old value if the
      user types a command accept as the first character%
      %------------------%
      LOCAL value;
      echon();
      CASE lookc() OF
```

```
      = CA: RETURN(oldvlu);
      NOT IN ['0, '9]: error();
      ENDCASE
         BEGIN
         value ← 0;
         WHILE lookc() IN ['0, '9] DO
            value ← value * 10 + (input() - '0);
         RETURN(value);
         END;
   END.

(typeno) PROCEDURE (number, astrng);  %type number on tty%
   %type the number which is passed on the tty if astrng is FALSE;
   otherwise put number in astrng.  Build the characters in the work
   area typnwa.  This routine can handle negative numbers also%
   %------------------%
   LOCAL worka, quotnt, remain, sign;
   REF astrng, worka;
   quotnt ← number;
   &worka ← $typnwa;
   DO
      BEGIN
      DIV quotnt/10, quotnt, remain;
      worka ← remain + '0;
      END
   UNTIL ((&worka ← &worka + 1) = $typend) OR (quotnt = 0);
   IF number < 0 THEN &worka ← '-
   ELSE BUMP DOWN &worka;
   DO
      IF &astrng THEN *astrng* ← *astrng*, worka
      ELSE
         BEGIN
         r1 ← worka;
         !JSYS pbout;
         END
   UNTIL (&worka ← &worka -1) < $typnwa;
   RETURN;
   END.

FINISH
```

TXCMND

```
<NLS>TXCMND.NLS;50, 3-JAN-72 16:44 MSC ; ["linkfg"];
FILE txcmnd   % L10   <REL-NLS>TXCMND.REL %

%.....Declarations.....%

   REF tda;
   REGISTER r1=1, r2=2, r3=3;

%.....Misc commands........%

   (tshptr) PROCEDURE(stid, cnt, astrng);
      REF astrng;
      *astrng* ← *astrng*, " = ";
      fechno(stid, &astrng);
      *astrng* ← *astrng*, " +", STRING(cnt);
      RETURN;
      END.

   (tpoint) PROCEDURE;
      todco('.);
      *stn* ← NULL;
      tshptr(tda.dacsp, tda.daccnt, $stn);
      typeas($stn);
      RETURN;
      END.

   (taltm) PROCEDURE;
      LOCAL ldel, rdel;
      LOCAL STRING treps[50];
      bl ← tda.dacsp;
      bl[1] ← tda.daccnt;
      CASE srchtype OF
         = word:
            BEGIN
            ldel ← '<;
            rdel ← '>;
            END;
         = contnt:
            BEGIN
            ldel ← '[;
            rdel ← ']';
            END;
         = contls:
            ldel ← rdel ← ';;
         ENDCASE error();
      *treps* ← SP, ldel, *conreg*, rdel;
      typeas($treps);
      typech(CA);
      specreg($conreg, srchtype, $bl);
      IF bl = endfil THEN error();
      mvcbl();
      RETURN;
      END.

   (tlinfed) PROCEDURE;
      LOCAL sw, stid;
```

```
   REF sw;
   &sw ←
      <SEQGEN, openseq>(getnxt(tda.dacsp), endfil, tda.davspec,
      tda.davspc2, tda.dausqcod, tda.dacacode);
   IF seqgen(&sw) = endfil THEN error();
   stid ← sw.swcstid;
   <SEQGEN, closeseq>(&sw);
   <TSPRT, printg>(stid, stid, stmtv);
   RETURN;
   END.

(taddr) PROCEDURE;
   todco(SP);
   ON SIGNAL
      = sighelp:
         BEGIN
         typeas($" ADDR CA");
         RETURN;
         END;
      ELSE;
   tbug($bl);
   mvcbl();
   RETURN;
   END.

(tua) PROCEDURE;
   LOCAL stid;
   todco('↑);
   IF (stid ← getbck(tda.dacsp)) = endfil THEN error();
   <TSPRT, printg>(stid, stid, stmtv);
   RETURN;
   END.

(tbslash) PROCEDURE;
   todco('\);
   tbsprnt(tda.dacsp);
   RETURN;
   END.

(tbsprnt) PROCEDURE(stid);
   % print the statement save and restore current CM %
   <TSPRT, movpr>(stid, stid, stmtv);
   RETURN;
   END.

(tslash) PROCEDURE(stid, cnt);
   LOCAL TEXT POINTER tcm, ptr;
   tcm ← ptr ← stid;
   tcm[1] ← cnt;
   ptr[1] ← MAX(1, tcm[1]-tslshchars);
   *lit* ← SP, ptr tcm, '<, LF, '>;
   FIND SE(tcm) ↑ptr;
   ptr[1] ← MIN(tcm[1]+tslshchars+1, ptr[1]);
   *lit* ← *lit*, tcm ptr;
   typeas($lit);
   RETURN;
```

```
   END.

(tcmnt) PROCEDURE;
   LOCAL char;
   todco('");
   WHILE (char ← input()) # CA DO todco(char);
      %echo comment characters%
   todco('");
   RETURN;
   END.

%.....Initialization and Error Routines.......%

(tstart)PROCEDURE;
   %-----------------%
   reset();
   END.

(reset) PROCEDURE;
   %-----------------%
   zap();
   STATE ← tstate, spec;
   mainct(); %start command parsing again%
   END.

(error) PROCEDURE;
   %-----------------%
   textfg ← FALSE;
   typech('?);
   clrbuf(0);  %clear todas input buffers %
   GOTO STATE;
   END.

%.....Primary Command Language I/O Routines...%

(tbug) PROCEDURE(ptr);
   REF ptr;
   ptr ← tda.dacsp;
   ptr[1] ← tda.daccnt;
   getadr(&ptr);
   RETURN END.

(getadr) PROCEDURE(ptr); %read statement address from tty%
   %parameter: address of tpointer=starting tpointer for
   relative addresses
   Returns: after having updated the tpointer
   Error conditions: calls error if receives unusual input
   %
   %-----------------%
   REF ptr;
   LOCAL TEXT POINTER oldptr;
   LOCAL STRING
      gder[50],  %for error string%
      gadstr[100]; % to hold the entire string for the address %
   (gad):
   IF tadlfg THEN linkfg ← FALSE;
```

```
    *gadstr* ← NULL;
    gadlit($gadstr);%collect the literal%
    oldptr ← ptr;
    oldptr[1] ← ptr[1];
    ON SIGNAL %catch any errors during interpretation%
       =gaderr:
          BEGIN
          *[MESSAGE]*←SP,*[MESSAGE]*,"?  ";
          typeas(MESSAGE);
          clrbuf(0);
          ptr ← oldptr;
          ptr[1] ← oldptr[1];
          GOTO gad;
          END;
       ELSE;
    gadgo($gadstr, $gder, &ptr, &tda); %interpret the string%
    tadlfg ← FALSE;
    RETURN END.

(gadlit) PROCEDURE(astrng); %read statement address from tty%
    %read a literal from the keyboard for getadr%
    %-----------------%
    LOCAL char, stid, litescape;
    REF astrng;
    echon();
    litescape ← todlit;
       %keep in local environment to avoid page fault to read on each
       character processed%
    CASE lookc() OF % see if the first character is a ? %
       = SP:
          BEGIN
          input();
          REPEAT CASE;
          END;
       = '?:
          BEGIN
          input();
          SIGNAL(sighelp);
          END;
       ENDCASE;
    LOOP CASE char ← input() OF
       = CD:  %command delete%
          GOTO STATE;
       = BC:  %backspace character%
          IF astrng.L > empty THEN
             BEGIN
             IF (char ← *astrng*[astrng.L]) = $ascalt THEN char ← '$;
             todco(char);
             bkc(&astrng);
             END;
       = BW:  %backspace word%
          BEGIN
          todco(BW);
          CASE *astrng*[astrng.L] OF
             = SP, = TAB, = CR:
                %space past invisible characters%
```

```
            IF astrng.L > empty THEN
               CASE bkc(&astrng) OF
                  = SP, = TAB, = CR: REPEAT 1;
                  ENDCASE;
         ENDCASE;
      LOOP CASE bkc(&astrng) OF
         =SP, =TAB, =CR, =ENDCHR: EXIT LOOP;
         ENDCASE;
      END;
   = $ascbst: %backspace statement%
      BEGIN
      *astrng* ← NULL;
      crlf();
      END;
   = $ctlr:   %retype literal%
      BEGIN
      crlf();
      typeas(&astrng);
      END;
   IN [SP,'←], IN ['a,'z], =TAB,=CR,=EOL,=LF,=litescape,=$ascalt:
      BEGIN
      CASE char OF
         =litescape: %literal escape character%
            BEGIN
            *astrng* ← *astrng*, char;
               %gadgo needs the escape char%
            char ← rawchr();
            END;
         =$ascalt: %altmode%
            todco('$);
         ENDCASE;
      *astrng* ← *astrng*, char;
      END;
   =CA, =C.: RETURN;
   ENDCASE
      dismes(2, $"*Illegal Character*");
END.

(gadgo) PROCEDURE(gadstr, gder, ptr, da);
   % called with a string that defines an address and a starting
   location %
   LOCAL char, count, cnt, oldcnt, scnbck;
   LOCAL STRING ggst[50];
   LOCAL TEXT POINTER pscan;
   REF gadstr, gder, ptr, da;
   count ← 1;
   cnt ← empty;
   LOOP
      BEGIN
      *gder* ← char ← *gadstr*[cnt ← cnt+1];
      CASE char OF
         =SP, =todlit:  % space means nothing %
            NULL;
         ='?: SIGNAL(sighelp);
         ='/:   % slash %
            IF nlmode # fulldisplay THEN tslash(ptr, ptr[1]);
```

```
='\:   % backslash %
   IF nlmode # fulldisplay THEN tbsprnt(ptr);
='S, ='s:  % succ %
   getpr($getsuc, count, &ptr);
='P, ='p:  % predecessor %
   getpr($getprd, count, &ptr);
='D, ='d:  % down %
   getpr($getsub, count, &ptr);
='U, ='u:  % up %
   getpr($getup, count, &ptr);
='H, ='h:  % head %
   getpr($gethed, count, &ptr);
='T, ='t:  % tail %
   getpr($getail, count, &ptr);
='E, ='e:  % end %
   BEGIN
   ptr ← <NLS, JUMP, getvnd>(ptr, da.davspec.vslev);
   ptr[1] ← 1;
   END;
='N, ='n:  % next %
   getpr($getnxt, count, &ptr);
='B, ='b:  % back %
   getpr($getbck, count, &ptr);
='R, ='r:  % return %
   IF count < 0 THEN
      BEGIN
      count ← -count;
      REPEAT('A);
      END
   ELSE
      BEGIN
      ptr ← getret(&da, count);
      ptr[1] ← 1;
      END;
='A, ='a:  % ahead %
   IF count < 0 THEN
      BEGIN
      count ← -count;
      REPEAT('R);
      END
   ELSE
      BEGIN
      ptr ← getahd(&da, count);
      ptr[1] ← 1;
      END;
='&: % Jump file return %
   gadjf(&ptr, $popls, &gder, count, &da);
='@: % Jump file ahead %
   gadjf(&ptr, $bumpls, &gder, count, &da);
='↑: % indirect  link %
   BEGIN
   oldcnt ← cnt;
   ptr[1] ← MAX(<NLS, FILMNP, fchtxt>(getsdb(ptr)), ptr[1]);
   <JUMP, lnkspec>(count, &ptr,
      $stno, $stn2, $stn, $num);
   IF stn2.L # empty THEN ptr ← <JUMP, nfstid> ($stn2, $stn)
```

```
    ELSE ptr ← <JUMP, ofstid> (&da, $stn);
    ptr[1] ← 1;
    IF num.L # empty THEN feedlt(&da, $num);
    IF tadlfg THEN linkfg ← -1;
    *gder* ← *gadstr*[oldcnt TO cnt];
    END;
='(:  % link %
    BEGIN
    oldcnt ← cnt;
    *ggst* ← '(;
    cnt ← gdlit2(&gadstr, cnt+1, $ggst, '));
    *ggst* ← *ggst*, ');
    FIND SF(*ggst*) ↑pscan;
    <JUMP, lnkspec>(1, $pscan,
       $stno, $stn2, $stn, $num);
    IF stn2.L # empty THEN ptr ← <JUMP, nfstid> ($stn2, $stn)
    ELSE ptr ← <JUMP, ofstid> (&da, $stn);
    ptr[1] ← count ← 1;
    IF num.L # empty THEN feedlt(&da, $num);
    IF tadlfg THEN linkfg ← -1;
    *gder* ← *gadstr*[oldcnt TO cnt];
    END;
='.:  % name %
    BEGIN
    oldcnt ← cnt;
    cnt ← gadname(&ptr, &gadstr, cnt+1, $ggst);
    *gder* ← *gadstr*[oldcnt TO cnt];
    END;
='::  % error %
    err($"Use .name instead of :name");
=$ascalt: % repeat last search %
    BEGIN
    oldcnt ← cnt;
    cnt ← gadfspec(&ptr, srchtype, &gadstr, cnt);
    UNTIL (count ← count-1) < 0 DO
       specreg($conreg, srchtype, &ptr);
    *gder* ← '$, *gadstr*[oldcnt+1 TO cnt];
    END;
='[: %content search%
    cnt ←
    conspt(&ptr,&gadstr,cnt+1,count,$ggst,&gder,contnt,']);
='<: %word search%
    cnt ←
    conspt(&ptr,&gadstr,cnt+1,count,$ggst,&gder,word,'>);
='; %semicolon% : %content search in statement%
    cnt ←
    conspt(&ptr,&gadstr,cnt+1,count,$ggst,&gder,contls,';);
='' %apostrophe% : %character search%
    BEGIN
    oldcnt ← cnt;
    IF *gadstr* [cnt ← cnt+1] = todlit THEN cnt ← cnt+1;
    *ggst* ← *gadstr* [cnt];
    *gder* ← *gadstr* [oldcnt TO cnt];
    specreg ($ggst, contnt, &ptr);
    IF ptr # endfil THEN
       BEGIN
```

UNTIL (count ← count-1) < 0 DO ✓

```
        *conreg* ← *ggst*;
        srchtype ← contnt;
        END;
    END;
='#: % marker %
    BEGIN
    oldcnt ← cnt := gdlit2(&gadstr, cnt+1, $ggst, SP);
    ptr ← <INPFBK, lkmkr>($ggst, da.dacsp.stfile : ptr[1]);
    *gder* ← *gadstr*[oldcnt TO cnt];
    END;
='+: %scan right%
    BEGIN
    count ← gadnum(&gadstr, cnt+1 : cnt);
    CASE *gadstr*[cnt+1] OF
        =SP, =ENDCHR: REPEAT 2 ('C);
        ENDCASE REPEAT LOOP;
    END;
='-: %scan left%
    BEGIN
    count ← -gadnum(&gadstr, cnt+1 : cnt);
    CASE *gadstr*[cnt+1] OF
        =SP, =ENDCHR: REPEAT 2 ('C);
        ENDCASE REPEAT LOOP;
    END;
IN ['0, '9]:
    BEGIN
    count ← gadnum(&gadstr, cnt : cnt);
    CASE *gadstr*[cnt+1] OF
        =SP, =ENDCHR: REPEAT 2 ('C);
        ENDCASE REPEAT LOOP;
    END;
='W, ='w: %word scan%
    BEGIN
    count ← gaddir(&ptr, count : scnbck);
    UNTIL (count ← count-1) < 0 DO
        FIND $LD $NLD;
    IF scnbck THEN FIND $LD;
    FIND ↑ptr;
    END;
='I, ='i: %invisible scan%
    BEGIN
    count ← gaddir(&ptr, count : scnbck);
    UNTIL (count ← count-1) < 0 DO
        FIND $NP $PT;
    IF scnbck THEN FIND $NP;
    FIND ↑ptr;
    END;
='V, ='v: %visible scan%
    BEGIN
    count ← gaddir(&ptr, count : scnbck);
    UNTIL (count ← count-1) < 0 DO
        FIND $PT $NP;
    IF scnbck THEN FIND $PT;
    FIND ↑ptr;
    END;
='C, ='c: %character scan%
```

```
            BEGIN
            count ← gaddir(&ptr, count);
            UNTIL (count ← count-1) < 0 DO FIND CH;
            FIND ↑ptr;
            END;
         ='←: %statement front%
            FIND SF(ptr) ↑ptr;
         ='>: %statement end%
            FIND SE(ptr) ↑ptr;
         =ENDCHR: %have finished interpreting the string%
            RETURN;
         ENDCASE
            SIGNAL(gaderr, &gder);
      count ← 1;
      IF ptr = endfil THEN SIGNAL(gaderr, &gder);
      IF POS ptr = SE(ptr) THEN FIND ptr < CH ↑ptr;
      END;  % of loop %
   END.
(conspt) PROCEDURE(ptr,astr1,cnt,count,astr2,astr3,type,stopchar);
   % used for content and word searches.  uses gadspt to extract and
   do the first execution. then repeats as many times as indicated
   by count. returns new value for cnt. %
   LOCAL oldcnt;
   REF astr1, astr3;
   oldcnt ← cnt := gadspt(ptr, &astr1, cnt, astr2, type, stopchar);
   UNTIL (count ← count-1) = 0 DO
      specreg(astr2, type, ptr);
   *astr3* ← *astr1*[oldcnt-1 TO cnt];
   RETURN(cnt) END.
(gadfspec) PROCEDURE(ptr, type, gadstr, cnt);
   % looks for F specification on searches %
   REF ptr, gadstr;
   CASE *gadstr*[cnt+1] OF
      =SP, =todlit:
         BEGIN
         BUMP cnt;
         REPEAT CASE;
         END;
      ='f, ='F:
         BEGIN
         BUMP cnt;
         IF type # contls THEN
            ptr.stpsid ← origin;
         ptr[1] ← empty;
         END;
      ENDCASE;
   RETURN(cnt) END.

(gadjf) PROCEDURE(ptr, proc, gder, count, da);
   LOCAL linkp, fileno, linkpf;
   REF ptr, proc, da;
   IF count < 0 THEN
      BEGIN
      &proc ← IF &proc = $popls THEN $bumpls ELSE $popls;
      count ← -count;
      END;
```

```
   linkpf ← da.dalink;
   UNTIL (count ← count-1) < 0 DO
      IF NOT(linkp ← da.dalink ← proc(&da)) THEN
         BEGIN
         da.dalink ← linkpf;
         SIGNAL(gaderr, gder);
         END;
   fileno ← jmpopn(/linkp/.lfilnm);
   restjs(linkp, fileno);
   ptr ← readjs(&da);
   ptr/1/ ← 1;
   IF tadlfg THEN linkfg ← linkp;
   da.dalink ← linkpf;
   RETURN;
   END.

(gadspt) PROCEDURE(ptr, astr1, cnt, astr2, type, stopchar);
   LOCAL char;
   REF ptr, astr1, astr2;
   *astr2* ← NULL;
   % collect the string %
   cnt ← gdlit2(&astr1, cnt, &astr2, stopchar);
   % look for F specification %
   cnt ← gadfspec(&ptr, type, &astr1, cnt);
   specreg(&astr2, type, &ptr);
   IF ptr # endfil AND type # name THEN
      BEGIN
      srchtype ← type;
      *conreg* ← *astr2*;
      END;
   RETURN (cnt) END.

(gadname) PROCEDURE(ptr, astr1, cnt, astr2);
   LOCAL TEXT POINTER tp1, tp2, tp3;
   REF ptr, astr1, astr2;
   tp1 ← <UTILTY, asrref>(&astr1);
   tp1/1/ ← cnt;
   <TXTEDT, nmdr>($tp1, $tp2, $tp3);
   *astr2* ← tp2 tp3;
   % look for F specification %
   cnt ← gadfspec(&ptr, name, &astr1, tp3/1/-1);
   specreg(&astr2, name, &ptr);
   RETURN(cnt) END.

(gaddir) PROCEDURE(ptr,count);
   REF ptr;
   IF count < 0 THEN
      BEGIN
      FIND ptr < ;
      RETURN ( -count, TRUE );
      END;
   FIND ptr > ;
   RETURN ( count, FALSE ) END.
(getpr) PROCEDURE(proc, count, ptr);
   REF proc, ptr;
   IF count < 0 THEN %do the inverse%
```

```
      BEGIN
      &proc ← CASE &proc OF
         = $getsuc : $getprd;
         = $getprd : $getsuc;
         = $getsub : $getup;
         = $getup  : $getsub;
         = $gethed : $getail;
         = $getail : $gethed;
         = $getnxt : $getbck;
         = $getbck : $getnxt;
         ENDCASE &proc;
      count ← -count;
      END;
   UNTIL (count ← count-1) < 0 DO
      BEGIN
      IF &proc = $getsuc AND getftl(ptr) THEN EXIT;
         % thus successor of statement with no successor is NOP %
      ptr ← proc(ptr);
      ptr[1] ← 1;
      END;
   RETURN END.
(gdlit2) PROCEDURE(astl, cnt, ast2, stopchar);
   % append characters to ast2 from astl starting at cnt until
   encounter a stopchar or the end of astl.  Return the count in astl
   of the stopchar. %
   LOCAL char;
   REF astl, ast2;
   LOOP
      BEGIN
      char ← *astl*[cnt];
      CASE char OF
         =ENDOHR, =stopchar:
            RETURN(cnt);
         =todlit:
            char ← *astl*[cnt ← cnt+1];
         ENDCASE;
      *ast2* ← *ast2*, char;
      BUMP cnt;
      END;
   END.
(gadnum) PROCEDURE(astr, cnt);
   % extract and evaluate number from string for gadgo.  Called with
   address of a string and a count specifying where to start looking
   for a number in the string.  Returns (value, cnt) where cnt gives
   the count which when incremented gives the character after the
   number. %
   LOCAL count, char;
   REF astr;
   IF (char ← *astr*[cnt]) NOT IN ['0, '9] THEN RETURN(1, cnt-1);
   count ← char - '0;
   WHILE (char ← *astr*[cnt ← cnt+1]) IN ['0, '9] DO
      count ← count*10 + char - '0;
   RETURN(count, cnt-1) END.

(tnumber)PROCEDURE(number);
   %Read a number for TNLS%
```

```
   %Return lentgh of number read%
   REF number;
   *number* ← NULL;
   WHILE input() IN ['0, '9] DO
      BEGIN
      *number* ← *number*, curchr;
      todco(curchr);
      END;
   RETURN(number.L);
   END.
(nqmlit) PROCEDURE(astrng, chrlst);
   % If next character is a '? then SIGNAL(sighelp,0) else equivalent
   to rdlit. %
   %-----------------%
   IF lookc() = '? THEN
      BEGIN
      input();
      SIGNAL(sighelp,0);
      END;
   RETURN(rdlit(astrng, chrlst)) END.
(rdlit) PROCEDURE (astrng, chrlst);  %read literal from keyboard%
   %read a literal from the keyboard, doing all of the control
   things, plus breaking on any control character identified by the
   list in chrlst% %returns 0 if buffer overflows, 1 if normal
   termination (ca or c.), 2 if a break character was input.%
   %-----------------%
   LOCAL char, stid;
   REF astrng;
   %assumes echon%
   LOOP
      BEGIN
      char ← input();
      IF chrlst > 0 AND chrpos(char, chrlst) THEN
         RETURN(2, char);
            %break character%
      CASE char OF
         = todlit:  %literal escape%
            BEGIN
            char ← rawchr();
            *astrng* ← *astrng*, char;
            END;
         = CD:  %command delete%
            GOTO STATE;
         = CA, = C.:  %command accept or center dot%
            RETURN(1, char);
         = BC:  %backspace character%
            IF astrng.L > empty THEN
               BEGIN
               todco(*astrng*[astrng.L]);
               bkc(&astrng);
               END;
         = BW:  %backspace word%
            BEGIN
            todco(BW);
            CASE *astrng*[astrng.L] OF
               = SP, = TAB, = CR:
```

```
                %space past invisible characters%
                IF astrng.L > empty THEN
                   CASE bkc(&astrng) OF
                      = SP, = TAB, = CR: REPEAT 1;
                      ENDCASE;
             ENDCASE;
          LOOP CASE bkc(&astrng) OF
             =SP, =TAB, =CR, =ENDCHR: EXIT LOOP;
             ENDCASE;
          END;
       = $ascbst: %backspace statement%
          BEGIN
          *astrng* ← NULL;
          todco(EOL);
          END;
       = $ctlr:   %retype literal%
          BEGIN
          crlf();
          typeas(&astrng)
          END;
       IN [SP, '←], IN ['a, 'z], =TAB, =CR, =EOL, =LF, =ALT:
          BEGIN
          IF astrng.L = astrng.M THEN
             RETURN(0, char);
                %astrng full--return error condition%
          *astrng* ← *astrng*, char;
          END;
       ENDCASE dismes(2, $"*Illegal Character*");
    END;
 END.

(txtlit) PROCEDURE(astrng); %read a literal into A-string%
   % This procedure reads a literal into the A-string whose
   address is passed.  It also checks for control-y (enter
   alte mode) and returns alterv as the completion code. %
   %-----------------%
   LOCAL rcndtn;
   echon(); %**for now call echon (later echolt)**turn off
   break on every character%
   IF (rcndtn ← rdlit(astrng, 0)) = 0 THEN err(6);
   echoff(); % turn break back on, echo off%
   RETURN (rcndtn);
   END.

(getctc) PROCEDURE;
   %scan for a ctc, and returns if ok. Otherwise goes to error%
   %-----------------%
   CASE input() OF
      =CA, =C., ='Y, ='y: RETURN;
      ENDCASE error();
   END.

(getvsp)  %get view specs from user%
   % this procedure accepts a string of viewspecs from the
   keyboard and then calls feedlt to put them into effect.  After
   calling this procedure, should call tresleV(stid) to have relative
```

```
   level made absolute wrt level of stid %
   %-----------------%
   PROCEDURE;
   *num* ← NULL;
   echon();
   LOOP
      CASE lookc() OF
         =SP:
            BEGIN
            input();
            REPEAT CASE;
            END;
         ='?:
            BEGIN
            input();
            SIGNAL(sighelp);
            END;
         ENDCASE EXIT;
   IF rdlit($num, 0) = 0 THEN err(6);
   IF num.L > empty THEN <JUMP, feedlt>(&tda, $num);
   RETURN;
   END.

(vspstatus)PROC(da, astr, actflag);
   %Puts status of current viewspecs in english into astring
   according to flag: flag = false, all viewspecs. flag = true, only
   viewspec which effect staement selection and are not standard%
   LOCAL vsp;
   REF astr, da;
   *astr* ← NULL;
   IF actflag THEN
      BEGIN
      END
   ELSE
      BEGIN
      END;
   RETURN(FALSE);
   END.

(treslev) PROCEDURE(stid);
   IF tda.davspec.vsrlev THEN
      tda.davspec ← <SEQGEN, reslev>(tda.davspec, getlev(stid));
   RETURN END.

(levadj) PROCEDURE (stid, astrng);
   % this procedure conducts dialogue with the user to do level
   adjusting (returns after updating astring), given an
   initial stid and string address. %
   %-----------------%
   LOCAL char;
   REF astrng;
   echoff();
   todco(SP);
   IF lvdjnm THEN
      BEGIN
      *stno* ← NULL;
```

```
        IF stid.stpsid = origin THEN fechnd(stid, $stno)
           ELSE fechnp(stid, $stno);
        typeas($stno);
        typech(SP);
        END;
     *astrng* ← NULL;
     LOOP
        BEGIN
        CASE (char ← lookc()) OF
           =CA, =C., =SP:
              BEGIN
              todco(input());
              EXIT LOOP;
              END;
           ='U, ='u, ='D, ='d, IN ['0, '9]:
              BEGIN
              *astrng* ← *astrng*, char;
              todco(char);
              END;
           ='?:
              BEGIN
              input();
              SIGNAL(sighelp);
              END;
           =BC:
              IF astrng.L > empty THEN
                 BEGIN
                 char ← *astrng*[astrng.L];
                 todco(BC);
                 todco(char);
                 astrng.L ← astrng.L - 1;
                 END;
           =BW, =$ascbst:
              BEGIN
              todco(char);
              *stn* ← NULL;
              END;
           =CD: GOTO STATE;
           ENDCASE EXIT LOOP;
           input();
        END;
     crlf();
     CASE char OF
        =CA, =C., =SP: NULL;
        ENDCASE todco(char);
     echon();
     RETURN;
     END.

%.....Support Routines.......%

   (clrbuf) %clear the todas input and (optionally) output
   buffers%
      %if outflg is non-zero the output buffer is also cleared.
      nothing is  returned%
      %-----------------%
```

```
   PROCEDURE (outflg);
      IF outflg THEN
         BEGIN
         %jsys to clear output buffer%
            r1 ← 101B;
            !JSYS cfobf;
         END;
      %jsys to clear input buffer%
         r1 ← 100B;
         !JSYS cfibf;
      buffct ← 0; %clear the todas buffer%
      RETURN;
      END.

(chrpos) PROCEDURE (char, astrng); %scan string for character%
   %chr in first parameter, a-string addr in second%
   %return TRUE if char in string, FALSE otherwise.%
   %-----------------%
   LOCAL tmpchr;
   REF astrng;
   CCPOS *astrng*;
   LOOP
      IF (tmpchr ← READC) = char THEN RETURN(TRUE)
      ELSE IF tmpchr = ENDCHR THEN RETURN(FALSE);
   END.

(astruc) PROCEDURE(astrng); %a-string to upper case%
   % convert a-string (in astrng) to upper case %
   %-----------------%
   LOCAL length, bytptr, char;
   REF astrng;
   IF (length ← astrng.L) = empty THEN RETURN(&astrng);
   bytptr ← chbptr(empty) + &astrng;
   DO IF (char ← ↑bytptr) IN ['a,'z] THEN
      .bytptr ← char - 40B
   UNTIL (length ← length - 1) = empty;
   RETURN(&astrng);
   END.

(mvcbl) PROCEDURE;
   mvcsp($bl);
   RETURN END.

(mvcsp) PROCEDURE(ptr);
   LOCAL fl;
   REF ptr;
   IF linkfg = -1 THEN %link return or ahead followed by link%
      BEGIN
      pushls(&tda, [fl←flntadr(ptr.stfile)].flastr);
      tda.dacsp ← ptr;
      tda.daempty ← FALSE;
      pushjs(&tda);
      mesfre(fl, ptr);
      END
   ELSE
      BEGIN
```

```
        IF linkfg#0 THEN %link return or ahead, no other links%
           BEGIN
           tda.dalink ← linkfg;
           setdpa(&tda, linkfg);
           fl ← flntadr(ptr.stfile);
           mesfre(fl, ptr);
           END;
        tda.dacsp ← ptr;
        IF readjs(&tda)#tda.dacsp THEN pushjs(&tda);
        END;
     tda.daccnt ← ptr[1];
     tadlfg ← TRUE;
     linkfg ← FALSE;
     RETURN;
     END.

(mesfre)PROCEDURE(fl, ptr);
     REF fl;
     dismes(2, fl.flastr);
     IF fl.fllock THEN lockmes(ptr.stfile);
     freflnt();
     RETURN;
     END.

%.....Help.....%

(tyhelp) PROCEDURE(type);
     CASE type OF
        =-1:  %type list of all commands%
           BEGIN
           *lit* ←
              EOL,
              "Append ", EOL,
              "Break ", EOL,
              "Copy ", EOL,
              "Delete ", EOL,
              "Execute ", EOL,
              "Fix ", EOL,
              "Goto ", EOL,
              "Insert ", EOL;
           *lit* ← *lit*,
              "Load ", EOL,
              "Move ", EOL,
              "Name delimiters ", EOL,
              "Output ", EOL,
              "Print ", EOL,
              "Replace ", EOL,
              "Substitute ", EOL,
              "Transpose ", EOL,
              "Update ", EOL,
              "Viewspecs ", EOL;
           *lit* ← *lit*,
              "Xset ", EOL,
              ". (show location)", EOL,
              "/ (print context)", EOL,
              "\ (print statement)", EOL,
```

```
                "↑ (move back and print)", EOL,
                "LF (line feed, move forward and print)", EOL,
                "SP ADDR CA (space, move to new address)", EOL,
                "ALT (altmode, repeat last search)";
            END;
        = 0:  %type the entities for edit commands%
            BEGIN
            *lit* ← EOL;
            tyhp(1);
            *lit* ← *lit*, EOL;
            tyhp(2);
            END;
        = 1, =2:  %type the structural or textual entities%
            BEGIN
            *lit* ← EOL;
            tyhp(type);
            END;
        ENDCASE;
    typeas($lit);
    RETURN END.

(tyhp) PROCEDURE(type);
    CASE type OF
        = 1:  %structural entities%
            *lit* ← *lit*,
                "Statement ", EOL,
                "Branch ", EOL,
                "Plex ", EOL,
                "Group ";
        = 2:  %textual entities%
            *lit* ← *lit*,
                "Character ", EOL,
                "Word ", EOL,
                "Number ", EOL,
                "Visible ", EOL,
                "Invisible ", EOL,
                "Link ", EOL,
                "Text ";
        ENDCASE;
    RETURN END.

FINISH of txcmnd
```

TXC1

```
<NIC-NLS>TXCT.NLS;8, 11-JAN-72 22:23 BLP ;
FILE txct  % L10  <NIC-NLS>TXCT.REL %

%.....Declarations.................%

   SET %for opcodes%
      LSH=242B, SETCM=460B;
   REGISTER r1=1, r2=2, r3=3, r4=4;
   REF tda;
   DECLARE forward=1, uselit=0;

%.....TNLS Execute commands parser........%

   (texecute) PROCEDURE;
      %------------------%
      LOCAL fileno, strtyp, jfn, filtyp, flags, errs, savsp1, savsp2;
      LOCAL TEXT POINTER stid, stid1, stid2;
      LOCAL STRING  str[400], flnam[100], levj[20];
      echo ($"Execute ");
      CASE inpcuc() OF
         = '?:
            help(
                 $"Assimilate ",
                 $"Browse ",
                 $"Catalog Numbers ",
                 $"Content Analyzer ",
                 $"Device Specification",
                 $"Edit ",
                 $"File Verify ",
                 $"Insert Sequential ",
                 $"Journal",
                 $"Marker ",
                 $"Name Delimiters ",
                 $"Quit",
                 $"Reset",
                 $"Status",
                 $"Unlock File ",
                 $"Viewchange",
                0);
         = 'A:  %assimilate%
            BEGIN
            echo ($"Assimilate");
            ON SIGNAL
               =sighelp:
                  BEGIN
                  *lit* ← EOL,
                     " (structure) S,B,G, or P (to) ADDR CA", EOL,
                     " (from) ADDR CA LEVADJ CA VIEWSPEC CA";
                  typeas($lit);
                  RETURN;
                  END;
               ELSE;
            prompt ($" Structure ");
            CASE inpcuc() OF
               ='S:
                  BEGIN
```

```
            echo($"Statement");
            strtyp ← stmtv;
            END;
         ='B:
            BEGIN
            echo($"Branch");
            strtyp ← brnchv;
            END;
         ='G:
            BEGIN
            echo($"Group");
            strtyp ← groupv;
            END;
         ='P:
            BEGIN
            echo($"Plex");
            strtyp ← plexv;
            END;
         =CD:  GOTO STATE;
         ='?:
            BEGIN
            tyhelp(1); %type structural entities%
            REPEAT;
            END;
      ENDCASE
            BEGIN
            typech('?);
            REPEAT;
            END;
   crlf();
   prompt ($"to ");
   tbug ($stid);
   prompt ($" from ");
   CASE strtyp OF
      =stmtv, =brnchv:
            BEGIN
            tbug ($stid1);
            stid2 ← stid1;
            END;
      =groupv:
            BEGIN
            tbug ($stid1);
            prompt($" ");
            tbug ($stid2);
            stid1 ← <STRMNP, grptst> (stid1, stid2 : stid2);
            END;
      =plexv:
            BEGIN
            tbug ($stid1);
            stid1 ← <STRMNP, plxset> (stid1 : stid2);
            END;
      ENDCASE error();
   levadj (stid, $levj);
   savsp1 ← tda.davspec;  %save viewspecs to restore later%
   savsp2 ← tda.davspc2;
   getvsp();
```

```
      treslev (stidl);
      IF strtyp = stmtv THEN tda.davspec.vslev ← getlev (stidl);
      xea (stid, stidl, stid2, $levj, tda.davspec, tda.davspc2,
      tda.dausqcod, tda.dacacode);
      tda.davspec ← savspl;  %restore viewspecs%
      tda.davspc2 ← savsp2;
      bl ← stid;
      bl[1] ← 1;
      mvcbl();
      freflnt();
      END;
   = 'B: %browse mode%
      BEGIN
      echo($"Browse ");
      CASE inpcuc() OF
         =CD: GOTO STATE;
         =CA, ='E:
            BEGIN
            echo($"Enter ");
            xebe(tda.dacsp.stfile);
            END;
         ='L :
            BEGIN
            echo($"Leave");
            echon();
            getctc();
            echo($" Keep Changes? ");
            echoff();
            xebl(answer(), tda.dacsp.stfile);
            END;
         = '?:
            BEGIN
            *lit* ← EOL,
               "Enter ", EOL,
               "Leave ";
            typeas($lit);
            RETURN;
            END;
         ENDCASE
            BEGIN
            typech('?);
            REPEAT;
            END;
      END;
   = 'C: %content analyser, Catalog Numbers%
      BEGIN
      echo($"C");
      CASE inpcuc() OF
         ='A: %Catalog Numbers%
            catnms();
         ='O: %Content Analyser%
            BEGIN
            echo($"ontent Analyzer ");
            ON SIGNAL
               =sighelp:
                  BEGIN
```

```
                typeas($" (at) ADDR CA");
                RETURN;
                END;
             ELSE;
          tbug($bl);
          IF (errs ← xgpc ($bl, &tda)) > 0 THEN
             BEGIN
             *str* ← STRING(errs), " error(s):  Type CA.";
             typeas ($str);
             clrbuf (0);  % clear input buffer %
             getctc();
             END;
          END;
       =sighelp:
          BEGIN
          *lit* ←EOL,
             "CAtalog Numbers", EOL,
             "COntent Analyzer ";
          typeas($lit);
          GOTO STATE;
          END;
       ENDCASE twohelp($"Catalog Numbers",  $"Content
       Analyser", 0);
    END;
= 'D: %Device Specification%
    BEGIN
    echo($"Device Specification");
    echo($" not implemented");
    END;
= 'E: %Execute Edit COMMAND%
    BEGIN
    echo($"Edit ");
    ON SIGNAL
       =sighelp:
          BEGIN
          *lit* ←  " (at) ADDR CA (CR) EDIT STUFF";
          typeas($lit);
          RETURN;
          END;
       ELSE;
    tbug($bl);
    editx($bl);
    tda.dacsp ← xrs(uselit, $bl, $bl, $lit);
    tda.daccnt ← 1;
    IF curchr = C. THEN trepstructure();
    RETURN;
    END;
= 'F:  %File Verify %
    BEGIN
    echo($"File ");
    todvfy();
       %goes to state%
    END;
= 'I: %insert sequential, identification system%
    BEGIN
    echo($"I");
```

```
CASE inpcuc() OF
   ='N: %insert sequential%
      BEGIN
      echo($"nsert Sequential from ");
      CASE inpcuc() OF
         ='A:
            BEGIN
            echo($"Assembler ");
            filtyp ← macfil;
            END;
         ='9:
            BEGIN
            echo($"940 ");
            filtyp ← n40fil;
            END;
         ='T:
            BEGIN
            echo($"Tenex ");
            filtyp ← tenfil;
            END;
         = '?:
            BEGIN
            *lit* ← EOL,
             "Assembler file " , EOL,
             "940 file " , EOL,
             "Tenex file";
            typeas($lit);
            END;
         =CD: GOTO STATE;
         ENDCASE
            BEGIN
            typech('?);
            REPEAT;
            END;
      prompt($" at ");
      ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA LEVADJ (from file)
            FILENAME");
            RETURN;
            END;
         ELSE;
      tbug($bl);
      levadj(bl, $levj);
      ON SIGNAL
         = ofilerr:
            BEGIN
            crlf();
            typeas (sysmsg);
            clrbuf (0);    % clear the input buffer %
            crlf();
            EXIT CASE;
            END;
         ELSE;
      prompt ($" from File ");
```

```
            echon();
            *flnam* ← NULL;
            rdlit ($flnam, 0);
            echoff();
            xei (bl, $levj, $flnam, filtyp);
            END;
         ='D: %identification system%
            BEGIN
            echo($"dentification submode");
            IF lookc() # CA THEN REPEAT CASE;
            input();
            <JIDENT, tjidcontrol>();
            END;
         =CD: GOTO STATE;
         ENDCASE
            BEGIN
            todco('?);
            REPEAT CASE;
            END;
      END;
   = 'J: %Execute Journal%
      BEGIN
      echo($"Journal ");
      crlf();
      tej();
      END;
   = 'M: %Marker%
      BEGIN
      echo($"Marker ");
      CASE inpcuc() OF
         ='F:
            BEGIN
            echo($"Fix");
            crlf();
            <TCMNDS, tf>();
            END;
         ='L, ='S:
            BEGIN
            echo($"List");
            crlf();
            <AUXCOD, seemkr>(tda.dacsp.stfile);
            END;
         ='R:
            LOOP BEGIN
            echo($"Release marker");
            prompt($" named ");
            *stn* ← NULL;
            echon();
            rdlit ($stn, $rnmdel);
            UNTIL stn.L MOD 5 = 0 DO *stn* ← *stn*, 0;
            <AUXCOD, delmkn>(lcfile(), stn[1]);
            IF curchr = CA THEN EXIT;
            crlf();
            END;
         ='?:
            BEGIN
```

```
            *lit* ← EOL,
               "Fix", EOL,
               "List", EOL,
               "Release";
            typeas($lit);
            RETURN;
            END;
         ENDCASE error();
      END;
   = 'N: %Name delimiter%
      BEGIN
      echo($"Name delimiter ");
      CASE inpcuc() OF
         = 'D: %Name delimiter display%
            BEGIN
            echo($"Display");
            ON SIGNAL
               =sighelp:
                  BEGIN
                  typeas($" (at) ADDR CA");
                  RETURN;
                  END;
               ELSE;
            LOOP
               BEGIN
               prompt($" at ");
               tbug($bl);
               mvcbl();
               <AUXCOD, getnmdls>(bl, $str);
               crlf();
               typeas($str);
               IF curchr # 0. THEN EXIT;
               crlf();
               END;
            END;
         = 'S: %Name delimiter statement%
            BEGIN
            echo ($"Statement");
            tens1 ($xens);
            END;
         = 'B: %Name delimiter branch%
            BEGIN
            echo ($"Branch");
            tens1 ($xenb);
            END;
         = 'P: %Name delimiter plex%
            BEGIN
            echo ($"Plex");
            tens1 ($xenp);
            END;
         = 'G: %Name delimiter group%
            BEGIN
            echo ($"Group");
            tens2 ($xeng);
            END;
         = '?:
```

```
      sequence), or a stastr, or the STID of a statement.
      Also the following fields of the work area are updated:
         SWCSTID (remains unchanged if an ENDFIL or stastr is the
         item returned)
         SWSTID (will be same as thing returned)
         SWSRSAV and SWMRSAV; SWCLVL; SWCALL, SWKFLG
         (unless conan code is using sport rather than using send or
         returning with flag set).
      It checks for a legal sequence work area.
      It also updates the statement vector if necessary.
      SEQGEN is actually merely a dispatch routine -- it calls either
      usqcod (user sequence code) or SSEQGEN (system sequence
      generator);
      It effects one-half of the coroutine linkage involved in
      port-sends (SPORT effects the other half).
      The input fork is informed that the sequence generator will
      handle rubouts.  Since the only way out of here is thru SPORT,
      the occurance of a rubout is checked for there and an ENDFIL
      sent if one occured.
      The stacks are switched here.  Thus no locals are allowed and
      the argument must be saved in a global. %

   REF sw;
   IF NOT (&sw IN [$sqgwas, $sqgaend]
      AND ((&sw - $sqgwas) MOD $sqwrkl) = 0 AND sw.swalloc) THEN
         SIGNAL (sqerr, $"illegal sequence work area");
   IF sw.swstid = endfil THEN RETURN (endfil);
   rubabt ← FALSE;  % tells input fork not to abort if see rubout %
   &sqsavsw ← &sw;       % save the argument in a global %
   % switch stacks %
      s ← sqsavsw.swsrsav := s;
      m ← sqsavsw.swmrsav := m;
   % depending on swcall (TRUE if last item in sequence was recieved
   by a normal return, FALSE it was recieved via a port-send), the
   next item in sequence is asked for by a CALL or a RETURN.  Note
   that OPENSEQ sets swcall TRUE and this is the only other place in
   NLS where it is tested or set. %
      IF sqsavsw.swcall := FALSE THEN
         BEGIN
         % call a user seqgen or SSEQGEN %
         IF sqsavsw.swusqcod AND sqsavsw.swvspec.vsusqf THEN
            [sqsavsw.swusqcod] (&sqsavsw, sqgnxt)
         ELSE sseqgen (&sqsavsw);
         sqsavsw.swcall ← TRUE;     % only get here if normal return
         made (not a port-send) %
         sport (&sqsavsw);     % SPORT will switch the stacks back and
         return to SEQGEN's caller %
         END
      ELSE RETURN;   % thru port, not to SEQGEN's caller %;
   END.

(sseqgen) PROCEDURE     % NOBODY BUT SEQGEN SHOULD CALL THIS ROUTINE %
   (sw);   % address of a sequence generator work area %

   % SSEQGEN
      Given address of sequence work area this procedure returns the
```

```
      next item in that sequence.
      SSEQGEN or SEQNXT or SEND or ucacod stores that value in SWSTID
      (and SWCSTID if it is not ENDFIL or a stastr).
      The routine SEQNXT finds the next STID, considering all
      viewspecs except content anlysis ones -- SSEQGEN takes care of
      those.
      Note that SEQNXT is not called the first time thru this
      routine.  Thus the first stid returned will be the one the
      sequence was initialized with -- unless conan code fails it.
      Rubouts are checked for after every call on conan code. %

   LOCAL stid;
   REF sw;
   LOOP
      BEGIN
      WHILE (sw.swvspec.vscapf OR (sw.swvspec.vscakf AND NOT
      sw.swkflg)) AND sw.swcacode > 0 AND sw.swstid # endfil DO
         % (conan must pass statements viewspec is on OR (turn off
         conan after first passed statement viewspec is on BUT
         haven't passed one yet)) AND there is a conan program AND we
         haven't reached the end of the sequence yet %
         BEGIN
         stid ← sw.swcstid;
         FIND SF(stid) ↑bl;
         [sw.swcacode] (&sw);
         IF flag OR inptrf %rubout% THEN
            BEGIN
            sw.swstid ← stid;  % a send from this same statement will
            have blown this field %
            sport (&sw);
            END;
         seqnxt (&sw);
         END;
      sport (&sw);
      seqnxt (&sw);
      END;
   END.

(seqnxt) PROCEDURE
   (sw);   % address of sequence work area %

   % SEQNXT
      Returns the next STID in this sequence (or ENDFIL if no more).
      It updates the following fields of the work area:
         swstid, swcstid (if not ENDFIL), and swclvl.
      It also updates the statement vector if necessary.
      It takes into account all viewspecs, except content analysis,
      during the search. %

   LOCAL stid, %current stid%
      vspec;   %first viewspec word (sw.swvspec)%
   REF sw;
   IF sw.swcstid.stastr THEN RETURN (sw.swstid ← endfil);
   vspec ← sw.swvspec;
   IF sw.swclvl >= vspec.vslev
      OR (stid ← getsub (sw.swcstid)) = sw.swcstid THEN
```

```
      BEGIN %see if superstructure fits viewspecs%
      IF (stid ← sw.swcstid) = sw.swlbstid
         OR vspec.vsbrof AND sw.swclvl <= sw.swslvl THEN
            RETURN (sw.swstid ← endfil);
      WHILE getftl (stid) DO
         BEGIN
         stid ← getsuc (stid);
         BUMP DOWN sw.swclvl;
         IF stid = sw.swlbstid
            OR vspec.vsbrof AND sw.swclvl <= sw.swslvl
            OR vspec.vsplxf AND sw.swclvl < sw.swslvl THEN
               RETURN (sw.swstid ← endfil);
         IF vspec.vsstnf THEN stvmod (sw.swsvw, up);
         END;
      IF stid.stpsid = origin THEN RETURN (sw.swstid ← endfil);
      %successor is next PSID in sequence%
      stid ← getsuc (stid);
      IF vspec.vsstnf THEN stvmod (sw.swsvw, suc);
      END
   ELSE
      BEGIN   %substructure fits viewspecs%
      BUMP sw.swclvl;   %increase level%
      IF vspec.vsstnf THEN stvmod (sw.swsvw, sub);
      END;
   RETURN (sw.swcstid ← sw.swstid ← stid);
   END.

(send) PROCEDURE
   (sw,    % address of a sequence work area %
   str);   % an ENDFIL or address of a string %

   % SEND
      Takes the place of the old send.
      Sets the swstid (but not the swcstid) field of the work area.
      Calls sport. %

   REF sw, str;
   IF &str = endfil THEN sw.swstid ← endfil
   ELSE
      BEGIN
      sw.swstid.stastr ← TRUE;
      sw.swstid.stadr  ← &str;
      IF str.L = empty THEN *str* ← SP; %no null strings permitted%
      END;
   sport (&sw);
   RETURN;
   END.

(sport)  PROCEDURE    % send-port mechanism %
   (sw);    % address of a sequence work area %

   % SPORT
      Effects one-half of the coroutine linkage involved in
      port-sends (SEQGEN effects the other half).
      Control o's and rubouts are checked for here.
      The stacks are switched here. %
```

```
    REF sw;
    IF NOT (&sw IN [$sqgwas, $sqgaend]
       AND ((&sw - $sqgwas) MOD $sqwrkl) = 0 AND sw.swalloc) THEN
          SIGNAL (sqerr, $"illegal sequence work area");
    sw.swkflg ← TRUE; % something has been returned in this sequence %
    % the stacks are about to be switched so save everything needed
    later in globals %
       &sqsavsw ← &sw;
    % switch stacks %
       s ← sqsavsw.swsrsav := s;
       m ← sqsavsw.swmrsav := m;
    IF inptrf THEN   % a contol o or a rubout has occurred %
       BEGIN
       rubabt ← TRUE;
       sqsavsw.swstid ← endfil;
       END;
    % the following RETURN has the effect of returning from the
    procedure last called from the now-in-effect stack %
    RETURN (sqsavsw.swstid);
    END.

(sqinit) PROCEDURE;   % sequence generator init %

    % SQINIT
       Is only called at INIT time.
       It initializes all the sequence work areas (including the
       attached stacks and sequence vector work areas). %

    LOCAL sw, swcnt;    % address of a sequence generator work area %
    REF sw;
    swcnt ← 0;
    FOR &sw ← $sqgwas UP $sqwrkl UNTIL >= $sqgaend DO
       BEGIN
       sw.swalloc ← FALSE;   % allocation bit %
       sw.swsvw ← $sqsvws + (swcnt * (svmxlev + 1));
       [$sqstks + (swcnt * $sqsksz)] ← $uflow;
       BUMP swcnt;
       END;
    RETURN;
    END.

(getsgw) PROCEDURE;   % get a sequence generator work area %

    % GETSGW
       Allocates, initializes some parts of, and returns the address
       of a sequence generator work area (a sequence work area, a
       statement vector work area, and a stack). %

    LOCAL sw, swcnt;    % address of a sequence generator work area %
    REF sw;
    swcnt ← 0;
    FOR &sw ← $sqgwas UP $sqwrkl UNTIL >= $sqgaend DO
       IF NOT sw.swalloc THEN
          BEGIN
          sw.swalloc ← TRUE;   % allocation bit %
```

```
            sw.swsrsav.rh ← sw.swmrsav.rh ← $sqstks + (swcnt * $sqsksz);
            sw.swsrsav.lh ← sw.swmrsav.lh ← -$sqsksz;
            RETURN (&sw);
            END
         ELSE BUMP swcnt;
      SIGNAL (sqerr, $"no more sequence work areas");
      END.

   (relsgw) PROCEDURE    % release a sequence generator work area %
      (sw);     % address of a sequence generator work area %

      % RELSGW
         Deallocates a sequence generator work area (a sequence work
         area, a statement vector work area, and a stack). %

      REF sw;
      IF NOT (&sw IN (.$sqgwas, $sqgaend)
         AND ((&sw - $sqgwas) MOD $sqwrkl) = 0 AND sw.swalloc) THEN
            SIGNAL (sqerr, $"illegal sequence work area");
      sw.swalloc ← FALSE;
      RETURN;
      END.

% SEQGEN, STATEMENT NUMBER / VECTOR / SIGNATURE UTILITY ROUTINES %

   (cpysw) PROCEDURE  % copy parts of one sequence work area to another%
      (fromsw, tosw);   % record pointers to work areas %
         % this has the effect of making the tosw point to the same item
         as the fromsw %
      REF fromsw, tosw;

      tosw.swcstid ← fromsw.swcstid;
      tosw.swstid  ← fromsw.swstid;
      tosw.swvspec ← fromsw.swvspec;
      tosw.swvsp2  ← fromsw.swvsp2;
      tosw.swsvw   ← fromsw.swsvw;
      tosw.swclvl  ← fromsw.swclvl;
      RETURN;
      END.

   (reslev) PROCEDURE(vspec, clevel);
      %Given a viewspec word and a level, this routine will perform any
      relative level adjustment required by the viewspec word and return
      the updated viewspec word.%
      IF vspec.vsrlev THEN
         BEGIN
         IF vspec.vslevd THEN % up %
            vspec.vslev ← MAX (clevel-vspec.vslev, 1)
         ELSE % down %
            vspec.vslev ← MIN (clevel+vspec.vslev, 63);
         vspec.vsrlev ← FALSE;
         vspec.vslevd ← FALSE;
         END;
      RETURN(vspec);
      END.
```

```
(stvect)
   %This routine generates a statement vector.
   It is called with a STID as the first argument and the
   address of the first word of a 17 word work area as the
   second.
      The number of words in the work area determine how many
      levels down in structure the routine can go (currently
      17).  ERR(5) is called if the work area is not long
      enough to contain the vector for the given STID.
      STVECT uses routine STPOS.
   STVECT returns an integer, which is the level of the
   statement.
   It also creates a statement vector.  Upon completion, the
   first word of the work area contains the level of the STID
   and subsequent cells contain the position within the
   respective plexes.  Thus for statement 1D2, the vector would
   contain (3, 1, 4, 2) in the first four cells.%
   %----------------%
   PROCEDURE (stid, stvwrk);
   LOCAL
      curwrd, %current word being built in vector%
      level;  %level of STID%
   REF stvwrk, curwrd;
   &curwrd ← &stvwrk + svmxlev;
   level ← 0;
   UNTIL stid.stpsid = origin DO %count up to origin%
      BEGIN
      IF (&curwrd ← &curwrd-1) <= &stvwrk THEN err(5);
      curwrd ← stpos(stid);
      stid ← getup(stid);
      BUMP level;
      END;
   stvwrk ← level;
   %move vector to top of work area%
   mvbfbf (&curwrd, &stvwrk+1, level);
   RETURN (level);
   END.


(stvmod)
   %This routine modifies a statement vector.  It is called
   with the address of a statement vector as the first
   argument, and a type parameter as the second.
   The statement vector is assumed to be initialized, and is
   modified as specified by TYPE.%
   %----------------%
   PROCEDURE (stvwrk, type);
   LOCAL word; %current word in statement vector%
   REF stvwrk, word;
   IF stvwrk = 0 THEN   % treat the origin statement specially %
      CASE type OF
         =sub:
            BEGIN
            stvwrk ← 1;
            [&stvwrk + 1] ← 1;
```

```
            END;
         =suc, =pred, =up: NULL;
         ENDCASE err(0)
   ELSE
      BEGIN
      &word ← &stvwrk + stvwrk;
      CASE type OF
         =sub:
            BEGIN
            IF (stvwrk ← stvwrk + 1) > svmxlev THEN err(5);
            /&word + 1/ ← 1;
            END;
         =suc:  word ← word + 1;
         =pred: word ← MAX (word-1, 1);
         =up:   stvwrk ← MAX (stvwrk-1, 0);
         ENDCASE err(0);
      END;
   RETURN;
   END.

(stpos)
   %Given a STID, this routine returns an integer which is
   the position of that statment within its plex.%
   %------------------%
   PROCEDURE (stid);
   LOCAL
      nstid,
      posit; %current position%
   IF getfhd(stid) THEN RETURN (1);
   posit ← 1;
   nstid ← gethed(stid);
   UNTIL nstid = stid DO
      BEGIN
      nstid ← getsuc(nstid);
      BUMP posit;
      END;
   RETURN (posit);
   END.


(getlev)
   %Called with STID, returns level of that statement.%
   %------------------%
   PROC(stid);
   LOCAL level; %current level%
   level ← 0;
   UNTIL stid.stpsid = origin DO
      BEGIN
      stid ← getup(stid);
      BUMP level;
      END;
   RETURN (level);
   END.


(fechno)
```

```
    %The statement number of a STID can be obtained by this
    routine.  Give the STID as the first argument, and the
    address of the A-string which is to contain the statement
    number as the second.  The statement number will be built
    in the A-string.  If the structure is not intact or the
    statement vector cannot be built, a call to RERROR or an
    EXCEED CAPICITY ERROR may result.%
    %-----------------%
    PROCEDURE(stid, astr);
    LOCAL fchsvw[50];
    stvect(stid, $fchsvw);
    fechnm($fchsvw, astr);
    RETURN;
    END.

(fechnp)
    %This is like FECHNO, but generates the statement number
    of the successor (whether it exists or not).%
    %-----------------%
    PROCEDURE(stid, astr);
    LOCAL fchsvw[50];
    stvect(stid, $fchsvw);
    stvmod($fchsvw, suc);
    fechnm($fchsvw, astr);
    RETURN;
    END.

(fechnd)
    %This is like FECHNO, but generates the statement number
    of the sub (whether it exists or not).%
    %-----------------%
    PROCEDURE(stid, astr);
    LOCAL fchsvw[50];
    stvect(stid, $fchsvw);
    stvmod($fchsvw, sub);
    fechnm($fchsvw, astr);
    RETURN;
    END.

(fechnm)
    %This routine will return the statement number, given the
    statement vector.  The address of the first word of the
    statement vector is provided as the first srgument, and
    the address of the A-string for the statement number is
    given as the second.
    The algorithm used is roughly as follows:
        The field of the statement vector corresponding to the
        current level is divided by a base.
            Two bases are used: 10 (for digits) and 27 (for
            letters and &).
        The quotient is added to an appropriate offset to create
        an ascii character, and this character is then added
        to the A-string.
        The division is repeated with the remainder until the
        quotient is less than the base. (This permits statement
        numbers of the form 12AB12D@.)
```

```
      The base is then changed and the statement vector field
      for the next level is then divided as above.%
   %-----------------%
   PROCEDURE(stvwrk, astr);
   LOCAL
      base, %current base%
      basnxt, % next base %
      curlev, %current level%
      posit, %current plex position%
      power, %power of base in plex postion = number of
         characters in statement number for this plex%
      char, %next character in statement number%
      offset, %converts number indicating position
         to number/alpha character%
      offnxt;  %next offset%
   REF astr, stvwrk;
   IF stvwrk = 0 THEN
      BEGIN
      *astr* ← *astr*, '0;
      RETURN;
      END;
   (base, basnxt, offset, offnxt) ←
      (numbase, alphbase, numoff, alphoff);
   curlev ← 0;
   LOOP
      BEGIN
      IF (curlev := curlev + 1) >= stvwrk THEN RETURN;
      posit ← stvwrk[curlev];
      %find largest power of base contained in plex position%
      power ← 1;
      WHILE power*base <= posit DO power ← power*base;
      %now put out the actual string%
      UNTIL power < 1 DO
         BEGIN
         DIV posit / power, char, posit;
         *astr* ← *astr*, char + offset;
         power ← power / base;
         END;
      (base, basnxt, offset, offnxt) ←
         (basnxt, base, offnxt, offset);
      END;
   END.


(fechux)
   %Given the address of an a-string as the first argument
   and a file number as the second, this routine returns the
   STID of the statement whose statement number is contained
   in the A-string.
   If the statement does not exist (or FECHUX is passed an
   illegal statement number) it returns a -1.
   The algorthm used is roughly as follows:
      The plex position indicated by each (alphabetic or
      numeric) level field in the statement number is converted
      to an integer, then the psid of the statement
      corresponding to that position is found.
```

```
   LODCHR is used to read characters from the statement
   number, and the resulting ascii character is converted
   to a number.
   This number is calculated by subtracting an appropriate
   offset (20B for numbers and 40B for lettrs), then
   multiplying the difference by an appropriate base (10
   for digits and 27 for letters).
   This number is added to any plex position already
   calculated.
   This subtraction, multiplication, and addition continues
   until a new level in the statement number is
   encountered.%
%-----------------%
PROCEDURE (astr, fileno);
LOCAL stid, %current stid%
   curlen, %position current character in statemet number%
   curc, %current character in statemeet number%
   base, %base for converting character to number%
   basnxt, %next base%
   zero, %zero point for converting character to number%
   zeronxt,  %next zero%
   posit; %current plex position%
REF astr;
posit ← 0;
stid ← origin;
stid.stfile ← fileno;
curlen ← 1;
base ← numbase;
basnxt ← alphbase;
zero ← numoff;
zeronxt ← alphoff;
LOOP
   BEGIN
   LOOP %read characters, calculate position%
      BEGIN
      curc ← *astr*[curlen];
      IF curc IN ['a, 'z] THEN curc ← curc - upcase;
      % check if have gone from alpha to digit or vice versa
      %
      IF (curc ← curc-zero) NOT IN [0, base) THEN EXIT;
      posit ← posit * base + curc;
      IF (curlen ← curlen+1) > astr.L THEN EXIT;
      END;
   %get stid in position indicated by posit, one level down
   from stid%
   IF posit = 0 THEN RETURN
      (IF stid.stpsid = origin THEN stid ELSE endfil);
   IF (stid := getsub(stid)) = stid THEN RETURN (endfil);
   UNTIL (posit ← posit-1) = 0 DO
      BEGIN
      IF getftl(stid) THEN RETURN (endfil);
      stid ← getsuc(stid);
      END;
   IF curlen > astr.L THEN RETURN (stid);
   (base, basnxt, zero, zeronxt, posit) ←
      (basnxt, base, zeronxt, zero, 0);
```

```
      END;
   END.


(fechsig)
   PROCEDURE (stid, astrng); %fetch statement signature%
   %assumes real statement--not an a-string%
   %----------------%
   LOCAL sdbadr, word, bytptr, count, char;
   REF sdbadr, astrng;
   % initials %
      word ← getint(stid);
      bytptr ← stbptr(empty) + $word;
      count ← 0;
      UNTIL (count ← count + 1) > 4 DO
         IF (char ← ↑bytptr) = 0 THEN EXIT LOOP
         ELSE *astrng* ← *astrng*, char;
      *astrng* ← *astrng*, SP;
   % date and time %
      lodsdb(getsdb(stid) : &sdbadr);
      <AUXCOD, dtfrmt>(sdbadr.stime, &astrng);
   RETURN;
   END.

FINISH of seqgen
```

SPLMNT

```
<NLS>SPLMNP.NLS;5, 1-JUN-71 17:52 WHP ;
FILE splmnp  % L10  <REL-NLS>SPLMNP %

   %...Statement property list procedures...%

      (copspl) PROCEDURE(source, dest);
         LOCAL ostdb, nstdb;
         IF (ostdb ← getspl(dest)) # 0 THEN deldl(ostdb);
         IF (ostdb ← getspl(source)) = 0 THEN RETURN;
         nstdb ← copdl(ostdb, dest.stfile);
         stoprop(dest, nstdb);
         RETURN END.

      (stoprop) PROCEDURE(stid, stdb);
         LOCAL b, n, ostdb, p;
         REF n, p;
         b ← lodsdb(stdb : &n);
         frzblk(b, 1);
         IF (ostdb ← getspl(stid)) = 0 THEN
            BEGIN
            stospl(stid, stdb);
            n.sback ← stid;
            n.sxflag ← FALSE;
            END
         ELSE
            LOOP
               BEGIN
               lodsdb(ostdb : &p);
               IF p.stype = n.stype THEN
                  BEGIN
                  indlrt(ostdb, stdb);
                  deldb(ostdb);
                  EXIT;
                  END;
               IF (ostdb.stpsdb ← p.sright) = 0 THEN
                  BEGIN
                  indlrt(ostdb, stdb);
                  EXIT;
                  END;
               END;
         frzblk(b, -1);
         RETURN END.

      (getprop) PROCEDURE(stid, type);
         LOCAL stdb, p;
         REF p;
         IF (stdb ← getspl(stid)) = 0 THEN RETURN(0);
         LOOP
            BEGIN
            lodsdb(stdb : &p);
            IF p.stype = type THEN RETURN(stdb);
            IF (stdb.stpsdb ← p.right) = 0 THEN RETURN(0);
            END;
         END.

      (delprop) PROCEDURE(stid, type);
```

```
      LOCAL stdb, p;
      REF p;
      IF (stdb ← getspl(stid)) = 0 THEN RETURN(0);
      LOOP
         BEGIN
         lodsdb(stdb : &p);
         IF p.stype = type THEN
            BEGIN
            deldb(stdb);
            RETURN;
            END;
         IF (stdb.stpsdb ← p.right) = 0 THEN RETURN;
         END;
      END.

%...Data list procedures...%

   (copdl) PROCEDURE(stdb, fileno);
      LOCAL nstdb, down, newdown, right, newright;
      nstdb ← copsdb(stdb, fileno);
      IF down ← getxdown(stdb) THEN
         BEGIN
         newdown ← copdl(down, fileno);
         stoxdown(nstdb, newdown);
         stoxback(newdown, nstdb);
         END;
      IF right ← getxright(stdb) THEN
         BEGIN
         newright ← copdl(right, fileno);
         stoxright(nstdb, newright);
         stoxback(newright, nstdb);
         END;
      RETURN(nstdb) END.

   (indldn) PROCEDURE(old, new);
      LOCAL down, tail, next;
      IF (down ← getxdown(old)) # 0 THEN
         BEGIN
         tail ← new;
         UNTIL (next ← getxright(tail)) = 0 DO tail ← next;
         stoxback(down, tail);
         stoxright(tail, down);
         END;
      stoxdown(old, new);
      stoxback(new, old);
      RETURN END.

   (indlrt) PROCEDURE(old, new);
      LOCAL right, tail, next;
      IF (right ← getxright(old)) # 0 THEN
         BEGIN
         tail ← new;
         UNTIL (next ← getxright(tail)) = 0 DO tail ← next;
         stoxback(right, tail);
         stoxright(tail, right);
         END;
```

```
      stoxright(old, new);
      stoxback(new, old);
      RETURN END.

   (deldl) PROCEDURE(stdb);
      LOCAL b, p, right;
      REF p;
      b ← lodsdb(stdb : &p);
      frzblk(b, 1);
      right ← stdb;
      WHILE (right.stpsdb ← p.sright) # 0 DO deldb(right);
      deldb(stdb);
      frzblk(b, -1);
      RETURN END.

%...Data branch procedures...%

   (copdb) PROCEDURE(stdb, fileno);
      LOCAL nstdb, down, ndown;
      nstdb ← copsdb(stdb, fileno);
      IF (down ← getxdown(stdb)) # 0 THEN
         BEGIN
         ndown ← copdl(down, fileno);
         indldn(nstdb, ndown);
         END;
      RETURN(nstdb) END.

   (remdb) PROCEDURE(stdb);
      LOCAL b, sdb, right, p;
      REF sdb, p;
      b ← lodsdb(stdb : &sdb);
      frzblk(b, 1);
      right ← stdb;
      IF (right.stpsdb ← sdb.sright) # 0 THEN
         BEGIN
         lodsdb(right : &p);
         p.sback ← sdb.sback;
         p.sxflag ← sdb.sxflag;
         END;
      IF sdb.sxflag THEN
         BEGIN
         stdb.stpsdb ← sdb.sback;
         stoxright(stdb, right);
         END
      ELSE
         stospl(makstid(stdb.stfile, sdb.sback), right);
      sdb.sback ← sdb.sright ← 0;
      frzblk(b, -1);
      RETURN END.

   (deldb) PROCEDURE(stdb);
      LOCAL down %etc% ;
      IF (down ← getxdown(stdb)) # 0 THEN deldl(down);
      remdb(stdb);
      % delete and consolidate as in fresdb now %
      END.
```

```
%...Statement data block procedures...%

   (copsdb) PROCEDURE(stdb, fileno);
      LOCAL n;
      REF n;
      %... make copy like now ...%
      n.sback ← n.sright ← n.sdown ← 0;
      n.sxflag ← TRUE;
      END.

FINISH of splmnp

   in gcol
   nstdb ← stdb;
   IF (nstdb.stpsdb ← p.sright) # 0 THEN
      stoxback(nstdb, new);
   IF (nstdb.stpsdb ← p.sdown) # 0 THEN
      stoxback(nstdb, new);
   IF p.sxflag THEN
      BEGIN
      nstdb.stpsdb ← p.sback;
      lodsdb(nstdb : &q);
      CASE old.stpsdb OF
         = q.sdown  : q.sdown ← new;
         = q.sright : q.sright ← new;
         ENDCASE badfil(fileno);
      END;
   ELSE stospl(makstid(fileno, p.sback), new);
```

SIR MNT

```
<NLS>STRMNP.NLS;18, 4-NOV-71 15:22 WHP ;
FILE strmnp    %   L10   <REL-NLS>STRMNP     %

vdeclarations%

   DECLARE  suc=1, sub=0;
   DECLARE  ctoff=FALSE, cton=TRUE;

%append statement routines%

   (apchk)
      %This routine checks that legal items have been chosen for append
      statement.  A statement cannot be appended to itself, nor can a
      statement be appended to a part of its own substructure.%
      %-------------%
      PROC(target, source);
      LOCAL srsub;
      IF target = source THEN err($"illegal append");
      IF (srsub ← getsub(source)) # source THEN
         movtst( target, srsub, getail(srsub));
      RETURN;
      END.

   (apdo)
      %This routine is called during the append statement process. It
      deletes the second statement specified after the text in the
      statement has been appended to the first.%
      %-------------%
      PROC(target, source, astring);
      LOCAL srsub;
      apchk(target, source);
      IF (srsub ← getsub(source)) # source THEN
         BEGIN
         *ladj* ← 'D;
         movplex( target, srsub, $ladj);
         END;
      <TXTEDT, staptx>(target, source, astring);
      delbranch(source);
      RETURN;
      END.

%break statement%

   (bkstat)
      %This routine gets a new stid, inserts it after OLDSTD, in the
      direction specified by DIR.  It then creates a new sdb for the
      statement using the tpointer and astring passed it.  It returns
      the value of the new stid it obtained.%
      %-------------%
      PROCEDURE(tptr, levstg, astring);
      LOCAL newstd; %new stid%
      REF tptr;
      newstd ← newrng(tptr.stfile);
      insgrp(tptr, newstd, newstd, levstg);
      <TXTEDT, brkst>(&tptr, newstd, astring);
      RETURN(newstd);
```

```
    END.

%primary control for structure manipulation%

    %copy%

        (copstat) PROC(target, source, levstg);
            LOCAL newsrc; %new stid%
            newsrc ← newrng(target.stfile);
            insgrp(target, newsrc, newsrc, levstg);
            copsdb(source, newsrc);
            RETURN(newsrc);
            END.

        (copbranch) PROCEDURE(target, source, levstg);
            LOCAL newsrc; %new stid%
            newsrc ← copgrp(target, source, source, levstg, ctoff);
            RETURN(newsrc);
            END.

        (copgroup) PROCEDURE(target, src1, src2, levstg);
            LOCAL newsrc; %new stid%
            src1 ← grptst(src1, src2 : src2);
            newsrc ← copgrp(target, src1, src2, levstg, ctoff);
            RETURN(newsrc);
            END.

        (copplex) PROCEDURE(target, source, levstg);
            LOCAL src1, %beginning of plex%
                src2, %end of plex%
                newsrc; %new stid%
            src1 ← plxset(source : src2);
            newsrc ← copgrp(target, src1, src2, levstg, ctoff);
            RETURN(newsrc);
            END.

    %delete%

        (xdele)
            %Given two stid's, this routine will delete the group bounded
            by them.  It deletes it from the ring, using REMGRP, and frees
            its SDB's and ring elements, using DELGRP.  It assumes that
            STID1, STID2 define a legal, ordered group.%
            %------------%
            PROCEDURE(stid1, stid2);
            LOCAL newsid;
            newsid ← getsuc(stid2);
            IF NOT remgrp(stid1, stid2) THEN err($"illegal delete");
            delgrp(stid1, stid2, newsid);
            RETURN;
            END.

        (delstat) PROCEDURE(stid);
            IF getsub(stid) # stid THEN err($"illegal delete");
            xdele(stid, stid);
            RETURN;
```

```
   END.

(delbranch) PROCEDURE(stid);
   xdele(stid, stid);
   RETURN;
   END.

(delgroup) PROCEDURE(stid1, stid2);
   stid1 ← grptst(stid1, stid2 :stid2);
   xdele(stid1, stid2);
   RETURN;
   END.

(delplex) PROCEDURE(stid);
   LOCAL stid1, %beginning of plex%
      stid2; %end of plex%
   stid1 ← plxset(stid :stid2);
   xdele(stid1, stid2);
   RETURN;
   END.

%insert%

(instat)
   %Given an stid, an astring, and a levstgection, this routine
   will insert the text in the astring as a statement after the
   stid passed, in the levstgection specified.  It returns the
   value of the new statement's stid.%
   %-------------%
   PROCEDURE(target, astring, levstg);
   REF astring;
   RETURN(inslit(target, &astring, levstg));
   END.

%move%

(xmove)
   %This routine assumes that SRC1 and SRC2 define the bounds of a
   legal, ordered group to be moved and that TARGET contains the
   target stid.  It checks that the target is not contained within
   the group.  If not, it removes the group from the structure, by
   calling REMGRP, and inserts it after TARGET, by calling
   INSGRP.%
   %-------------%
   PROCEDURE(target, src1, src2, levstg);
   LOCAL newsrc; %new stid (end of group moved)%
   movtst(target, src1, src2);
   IF target.stfile = src1.stfile THEN
      BEGIN %intrafile move%
      IF NOT remgrp(src1, src2) THEN err($"illegal move");
      insgrp(target, src1, src2, levstg);
      newsrc ← src2;
      END
   ELSE
      BEGIN %cross file move%
      newsrc ← copgrp(target, src1, src2, levstg, cton);
```

```
      xdele(src1, src2);
      END;
   RETURN(newsrc);
   END.

(movstat) PROCEDURE(target, source, levstg);
   IF getsub(source) # source THEN err($"illegal move");
   RETURN( xmove(target, source, source, levstg));
   END.

(movbranch) PROCEDURE(target, source, levstg);
   RETURN( xmove(target, source, source, levstg));
   END.

(movgroup) PROCEDURE(target, src1, src2, levstg);
   src1 ← grptst(src1, src2 : src2);
   RETURN( xmove(target, src1, src2, levstg));
   END.

(movplex) PROCEDURE(target, source, levstg);
   LOCAL src1, %beginning of plex%
      src2; %end of plex%
   src1 ← plxset(source :src2);
   RETURN( xmove(target, src1, src2, levstg));
   END.

%replace%

(xrepl)
   %This routine replaces the group delimited by TGT1,2 by that
   delimited by SRC1,2.  First it copies SRC1,2 after TGT1,2; then
   it deletes TGT1,2.  (It assumes that the stid's define legal,
   ordered groups.)%
   %------------%
   PROCEDURE(tgt1, tgt2, src1, src2);
   LOCAL newsrc; %new stid (end of group)%
   newsrc ← copgrp(tgt2, src1, src2, sucdir, ctoff);
   xdele(tgt1, tgt2);
   RETURN(newsrc);
   END.

(repstat) PROCEDURE(bugdit, target, source, astring);
   REF astring;
   dsttx(target); %free SDB%
   IF bugdit THEN
      copsdb(source, target)
   ELSE %literal input%
      ST target ← *astring*;
   RETURN(target);
   END.

(repbranch) PROCEDURE(bugdit, target, source, astring);
   IF bugdit THEN
      RETURN ( xrepl (target, target, source, source))
   ELSE RETURN ( rpllit (target, target, astring));
   END.
```

```
(repgroup)
   PROCEDURE(bugdit, tgt1, tgt2, src1, src2, astring);
   tgt1 ← grptst(tgt1, tgt2 :tgt2);
   IF bugdit THEN
      BEGIN
      src1 ← grptst(src1, src2 :src2);
      RETURN( xrepl(tgt1, tgt2, src1, src2));
      END
   ELSE RETURN(rpllit(tgt1, tgt2, astring));
   END.

(repplex) PROCEDURE(bugdit, target, source, astring);
   LOCAL
      src1, %beginning of plex to replace tgt1%
      src2, %end of plex to replace tgt2%
      tgt1, %beginning of plex to be replaced%
      tgt2; %end of plex to be replaced%
   tgt1 ← plxset(target : tgt2);
   IF bugdit THEN
      BEGIN
      src1 ← plxset(source : src2);
      RETURN( xrepl(tgt1, tgt2, src1, src2));
      END
   ELSE RETURN(rpllit(tgt1, tgt2, astring));
   END.

%transpose%

(xtran)
   %This routine transposes the groups defined by FIRST1 and
   FIRST2 (first group) and SECND1 and SECND2 (second group).  The
   first group need not be ahead of the second in the file
   structure, but the routine assumes that FIRST1,2 and SECND1,2
   define legal groups.  It checks to see that the groups don't
   overlap.
      If one group follows the other immediately, then the second
      group is merely deleted from the structure and inserted
      after the pred (or source) of the first; the first the is in
      the right position automatically.  Otherwise, the routine
      finds the pred (or source) of each group.
         If the two groups are in the same file, it calls REMGRP
         to delete one group from the structure, then INSGRP to
         insert the group after the pred (or source) of the other
         group.  The same delete/insert sequence is followed for
         the other group.  Otherwise, it uses COPGROUP twice and
         XDELE twice to perform the requisite operations.%
   %-------------%
   PROCEDURE(first1, first2, secnd1, secnd2);
   LOCAL
      gp1tgt, %target for group delimited by first1,first2%
      gp1dir, %direction for insertion of above group%
      gp2tgt, %target for group delimited by secnd1,secnd2%
      gp2dir; %direction for insertion of above group%
   trntst(first1, first2, secnd1, secnd2);
   IF getfhd(secnd1) THEN
```

```
        BEGIN
        gpltgt ← getup(secnd2);
        gpldir ← down;
        END
    ELSE
        BEGIN
        gpldir ← sucdir;
        IF (gpltgt ← getprd(secnd1)) = first2 THEN
            BEGIN %SECND1,2 follows FIRST1,2 directly%
            IF NOT remgrp(first1, first2) THEN
                err($"illegal transpose");
            insgrp(secnd2, first1, first2, sucdir);
            RETURN;
            END;
        END;
    IF getfhd(first1) THEN
        BEGIN
        gp2tgt ← getup(first2);
        gp2dir ← down;
        END
    ELSE
        BEGIN
        gp2dir ← sucdir;
        IF (gp2tgt ← getprd(first1)) = secnd2 THEN
            BEGIN %FIRST1,2 follows SECND1,2 directly%
            IF NOT remgrp(secnd1, secnd2) THEN
                err($"illegal transpose");
            insgrp(first2, secnd1, secnd2, gp2dir);
            RETURN;
            END;
        END;
    IF first1.stfile = secnd1.stfile THEN
        BEGIN %intrafile transpose%
        IF NOT remgrp(secnd1, secnd2) THEN
            err($"illegal transpose");
        insgrp(gp2tgt, secnd1, secnd2, gp2dir);
        IF NOT remgrp(first1, first2) THEN
            err($"illegal transpose");
        insgrp(gpltgt, first1, first2, gpldir);
        END
    ELSE
        BEGIN %cross file transpose%
        copgrp(gpltgt, first1, first2, gpldir, cton);
        copgrp(gp2tgt, secnd1, secnd2, gp2dir, cton);
        xdele(first1, first2);
        xdele(secnd1, secnd2);
        END;
    RETURN;
    END.

(trntst)
    %This routine checks that the groups specified by the transpose
    commands have legal bounds.  It does this by calling MOVTST to
    see if FIRST1,2 are in the group bounded by SECND1,2 or
    SECND1,2 in the group bounded by FIRST1,2.%
    %------------%
```

```
   PROCEDURE(first1, first2, secnd1, secnd2);
   movtst(first1, secnd1, secnd2);
   movtst(first2, secnd1, secnd2);
   movtst(secnd1, first1, first2);
   movtst(secnd2, first1, first2);
   RETURN;
   END.

(trstat)
   PROCEDURE(stid1, stid2);
   LOCAL
      psdb1, %psdb of stid1%
      psdb2, %psdb of stid2%
      newsd1, %new stid, replaces stid1 in cross-file
      trpose%
      newsd2, %new stid, replaces stid2 in cross-file
      trpose%
      tmpstd, %temporary to save stid%
      sdbadr; %adress of sdb%
   REF sdbadr;
   IF stid1.stfile = stid2.stfile THEN
      BEGIN  %intrafile transpose%
      psdb1 ← getsdb(stid1);  psdb2 ← getsdb(stid2);
      lodsdb(psdb1 :&sdbadr); sdbadr.spsid ← stid2.stpsid;
      stosdb(stid2, psdb1);
      lodsdb(psdb2 :&sdbadr); sdbadr.spsid ← stid1.stpsid;
      stosdb(stid1, psdb2);
      END
   ELSE
      BEGIN %cross file transpose%
      newsd1 ← copstat(stid1, stid2, sucdir);
      upctbl(stid2, newsd1, stid2);
      newsd2 ← copstat(stid2, stid1, sucdir);
      upctbl(stid1, newsd2, stid1);
      IF (tmpstd ← getsub(stid1)) # stid1 THEN
         movplex(newsd1, tmpstd, down);
      xdele(stid1, stid1);
      IF (tmpstd ← getsub(stid2)) # stid2 THEN
         movplex(newsd2, tmpstd, down);
      xdele(stid2, stid2);
      END;
   RETURN;
   END.

(trbranch) PROCEDURE(first, second);
   xtran(first, first, second, second);
   RETURN;
   END.

(trgroup) PROCEDURE(first1, first2, secnd1, secnd2);
   first1 ← grptst(first1, first2 : first2);
   secnd1 ← grptst(secnd1, secnd2 : secnd2);
   xtran(first1, first2, secnd1, secnd2);
   RETURN;
   END.
```

```
   (trplex) PROCEDURE(first, second);
      LOCAL
         firstl, %beginning of first plex%
         first2, %end of first plex%
         secndl, %beginning of second plex%
         secnd2; %end of second plex%
      firstl ← plxset(first : first2);
      secndl ← plxset(second : secnd2);
      xtran(firstl, first2, secndl, secnd2);
      RETURN;
      END.

%xset%

   (xstruc)
      %Given two stid's, this routine will XSET each statement in the
      group bounded by them.  It assumes that STIDl, STID2 define a
      legal, ordered group.%
      %-------------%
      PROCEDURE(stidl, stid2);
      grpapply(stidl, stid2, $cshftall);
      RETURN;
      END.

   (xsetbranch) PROCEDURE(stid);
      xstruc(stid, stid);
      RETURN;
      END.

   (xsetgroup) PROCEDURE(stidl, stid2);
      stidl ← grptst(stidl, stid2 :stid2);
      xstruc(stidl, stid2);
      RETURN;
      END.

   (xsetplex) PROCEDURE(stid);
      LOCAL stidl, %beginning of plex%
         stid2; %end of plex%
      stidl ← plxset(stid :stid2);
      xstruc(stidl, stid2);
      RETURN;
      END.

%assimilate%

   (xassim) PROCEDURE
      (target, srcl, src2, levstg, vspecl, vspec2, usqcod, cacode);
      LOCAL newtgt, newsrc, lstlev, toplev, sqgwrk;
      REF levstg, sqgwrk;
      &sqgwrk ← openseq (srcl, src2, vspecl, vspec2, usqcod, cacode);
      IF (newsrc ← seqgen(&sqgwrk)) # endfil THEN
         BEGIN
         newtgt ←
            IF newsrc.stastr THEN
               instat (target, newsrc.stadr, &levstg)
            ELSE copstat(target, newsrc, &levstg);
```

```
            toplev ← lstlev ←
               IF sqgwrk.swclvl = 0 THEN 1 ELSE sqgwrk.swclvl;
            WHILE (newsrc ← seqgen(&sqgwrk)) # endfil DO
               BEGIN
               *levstg* ← NULL;
               CASE sqgwrk.swclvl OF
                  =lstlev: NULL;
                  >lstlev:
                     BEGIN
                     *levstg* ← 'D;
                     lstlev ← lstlev + 1;
                     END;
                  <lstlev:
                     WHILE lstlev > toplev DO
                        BEGIN
                        *levstg* ← *levstg*, 'U;
                        IF ((lstlev ← lstlev - 1) = sqgwrk.swclvl) THEN
                           EXIT LOOP 1;
                        END;
                  ENDCASE NULL;
               newtgt ←
                  IF newsrc.stastr THEN
                     instat (newtgt, newsrc.stadr, &levstg)
                  ELSE copstat (newtgt, newsrc, &levstg);
               END;
            END;
         closeseq (&sqgwrk);
         RETURN;
         END.

%structural inserts%

   (newsuc)
      %This routine gets a new stid and then inserts it as the suc of
      the stid passed it.  It returns the new stid.%
      %-------------%
      PROCEDURE(stid);
      LOCAL newstd; %new stid%
      newstd ← newrng(stid.stfile);
      inss(stid, newstd, newstd);
      RETURN(newstd);
      END.

   (newsub)
      %This routine gets a new stid and then inserts it as the sub of
      the stid passed it.  It returns the new stid.%
      %-------------%
      PROCEDURE(stid);
      LOCAL newstd; %new stid%
      newstd ← newrng(stid.stfile);
      insd(stid, newstd, newstd);
      RETURN(newstd);
      END.

   (insgrp)
      %Insert SRC1, SRC2 at TARGET according to DIR.  If DIR is true,
```

```
   then the group is inserted as the successor; otherwise it is
   inserted as the sub.  (SRC1 and SRC2 are assumed to define a
   legal, ordered group.)%
   %-------------%
   PROCEDURE(target, src1, src2, levstg);
   LOCAL dir;
   REF levstg;
   IF levstg.L # empty THEN target ← levset(target, &levstg : dir)
   ELSE dir ← suc;
   IF target.stpsid # origin AND dir THEN
      inss(target, src1, src2)
   ELSE insd(target, src1, src2);
   RETURN;
   END.

(inss)
   %Given an stid, this routine inserts a group, defined by the
   second and third arguments, as the suc of the first argument.
   First it makes the suc of STID the suc of GRP2, and updates the
   tail flag; then it makes GRP1 the suc of STID and updates the head
   and tail flags.%
   %-------------%
   PROCEDURE(stid, grp1, grp2);
   IF stid.stfile # grp1.stfile THEN err($"illegal insert");
   IF grp1.stfile # grp2.stfile THEN err($"illegal group");
   stosuc(grp2, getsuc(stid));
   stoftl(grp2, getftl(stid));
   stosuc(stid, grp1);
   stofhd(grp1, FALSE);
   stoftl(stid, FALSE);
   RETURN;
   END.

(insd)
   %Given an stid, this routine inserts the group defined by the
   second and third arguments down from the first argument.%
   %-------------%
   PROCEDURE(stid, grp1, grp2);
   LOCAL substd; %stid of sub of stid passed as argument%
   IF stid.stfile # grp1.stfile THEN err($"illegal insert");
   IF grp1.stfile # grp2.stfile THEN err($"illegal group");
   IF (substd ← getsub(stid)) # stid THEN
      BEGIN
      stofhd(substd, FALSE);
      stoftl(grp2, FALSE);
      END
   ELSE stoftl(grp2, TRUE);
   stosuc(grp2, substd);
   stosub(stid, grp1);
   stofhd(grp1, TRUE);
   RETURN;
   END.

(levset)
   %Combines STID and DIR to give final target psid for level
   specification during structural editing.  If the new statement is
```

```
      to be the successor of the stid returned, DIR will be TRUE upon
      completion of this routine, otherwise the new statement will be
      inserted as the down of the stid returned.%
      %---------------%
      PROCEDURE(stid, levstg);
      LOCAL dir, count, numb;
      REF levstg;
      dir ← 0;
      IF levstg.L # empty THEN
         BEGIN
         count ← 1;
         DO
            CASE *levstg*[count] OF
               IN ['0, '9]:
                  BEGIN
                  numb ← *levstg*[count] - '0;
                  BUMP count;
                  CASE *levstg*[count] OF
                     ='d, ='D: dir ← dir + numb;
                     ='u, ='U, =ENDCHR: dir ← dir - numb;
                     ENDCASE;
                  END;
               ='U, ='u: BUMP DOWN dir;
               ='D, ='d: BUMP dir;
               ENDCASE NULL
         UNTIL (count ← count + 1) > levstg.L;
         END;
      CASE dir OF
         =0: BUMP dir; %successor%
         >0: dir ← 0 %down%
         ENDCASE %up number of levels indicated by dir%
      WHILE (dir ← dir + 1) <= 0
         DO stid ← <STR10, getup>(stid);

      RETURN(stid, dir);
      END.

%insert and replace using literal register%

   (inslit)
      %This routine gets a new stid, inserts it after the stid passed
      it, in the direction specified, and copies the A-string passed it
      into the statement's SDB.  It returns the value of the new stid.%
      %-------------%
      PROCEDURE(target, astring, dir);
      LOCAL newstid; %new stid%
      REF astring;
      newstid ← newrng(target.stfile);
      insgrp(target, newstid, newstid, dir);
      ST newstid ← *astring*;
      RETURN(newstid);
      END.

   (rpllit)
      %This routine replaces the group defined by the first two
      arguments passed it with a new statement containing the text in
```

```
      the astring passed it.  It returns the value of the new stid.
         First it gets a new stid, and inserts it as the successor of
         GRP2.  It then deletes the group bounded by GRP1, GRP2, and
         puts the text in the astring passed it in the new SDB.%
      %-------------%
      PROCEDURE(grp1, grp2, astring);
      LOCAL stid; %stid for new statement%
      REF astring;
      IF grp1.stfile # grp2.stfile THEN err($"illegal group");
      stid ← newrng(grp1.stfile);
      inss(grp2, stid, stid);
      xdele(grp1, grp2);
      ST stid ← *astring*;
      RETURN(stid);
      END.

%test and set bounds of structure%

   (plxset)
      %Given a stid, this routine will return the stid's of the head and
      tail, respectively, of the plex in which the stid appears.%
      %-------------%
      PROCEDURE(stid);
      LOCAL grp1, %stid of head%
         grp2; %stid of tail%
      IF stid.stpsid = origin THEN grp1 ← grp2 ← stid
      ELSE
         BEGIN
         grp1 ← gethed(stid);
         grp2 ← getail(stid);
         END;
      RETURN(grp1, grp2);
      END.

   (grptst)
      %Given two stid's, this routine checks that they specify a legal
      group; it also returns them ordered (GRP1,GRP2). If the stid's do
      not form a legal group, an err($"illegal group") is issued.%
      %-------------%
      PROCEDURE(stid1, stid2);
      LOCAL t1, %working stid1%
         t2; %working stid2%
      IF stid1.stpsid = origin OR stid2.stpsid = origin
      OR stid1.stfile # stid2.stfile THEN err($"illegal group");
      t1 ← stid1; t2 ← stid2;
      LOOP
         BEGIN %is stid2 on same level, and after, stid1%
         IF t1 = stid2 THEN RETURN(stid1, stid2);
         IF getft1(t1) THEN LOOP
            BEGIN %is stid1 on same level after stid2%
            IF t2 = stid1 THEN RETURN(stid2, stid1);
            IF getft1(t2) THEN err($"illegal group");
            t2 ← getsuc(t2);
            END;
         t1 ← getsuc(t1);
         END;
```

```
    END.

(movtst)
    %Given an stid as the first argument, this routine makes sure that
    the stid is not in the group bounded by the second and third
    arguments passed it.  If it is, it does an err.
       First it sees whether STID is a suc of GRP2 on the same level.
       If not, it then sees whether STID's sucs (and ups) are GRP1 or
       GRP2.%
    %-------------%
    PROCEDURE(stid, grp1, grp2);
    LOCAL tmpsid; %working stid%
    IF grp2.stpsid = origin THEN err($"illegal move");
    IF grp1.stfile # grp2.stfile THEN err($"illegal group");
    IF stid.stfile # grp1.stfile
       OR stid.stpsid = origin THEN RETURN;
    IF stid = grp1 OR stid = grp2 THEN err($"illegal move");
    tmpsid ← grp2;
    LOOP %is stid in plex after grp2%
       BEGIN
       IF getftl(tmpsid) THEN
          BEGIN
          tmpsid ← getsuc(tmpsid);
          EXIT;
          END;
       tmpsid ← getsuc(tmpsid);
       IF stid = tmpsid THEN RETURN; %psid in plex after grp2%
       END;
    LOOP %see if grp1, grp2 in superstructure of psid%
       BEGIN
       IF stid = grp1 THEN err($"illegal move");
       IF stid.stpsid = origin OR stid = tmpsid THEN RETURN;
       LOOP
           BEGIN
           IF stid = grp2 THEN err($"illegal move");
           IF getftl(stid) THEN EXIT;
           stid ← getsuc(stid);
           IF stid = grp1 THEN RETURN;
           END;
        stid ← getsuc(stid);
        END;
    END.

%basic manipulation routines%

(grpapply)
    %Assumes that stid1 and stid2 form a legal group with stid1 the
    first statement and stid2 the second.  For each statement in the
    group, calls proc(stid).%
    %-----------------%
    PROCEDURE(stid1, stid2, proc);
    LOCAL stid;  REF proc;
    stid ← stid1;
    LOOP
       BEGIN
       proc(stid);
```

```
      % go down the tree %
      WHILE (stid := getsub(stid)) # stid DO proc(stid);
      % have reached a statement with no substructure %
      LOOP
         BEGIN
         % at this point have applied proc to stid and all of stid's
         substructure %
         IF stid = stid2 THEN RETURN;
         IF stid = stid1 THEN
            BEGIN
            stid1 ← stid ← getsuc(stid);
            EXIT LOOP;
            END;
         IF NOT getft1(stid := getsuc(stid)) THEN EXIT LOOP;
         END;
      END;
   END.


(copgrp)
   %Given an stid as the first argument, this routine copies the
   group bounded by the second and third arguments after the first
   stid, in the direction specified by the fourth argument.  The
   fifth argument indicates whether the correspondence list should be
   updated.
      It proceeds by constructing new ring elements for each branch
      defined by the top-level statements in the group. Then it
      copies these data blocks into freshly allocated SDB's.  It then
      inserts the newly created group after the stid passed it.%
   %-------------%
   PROCEDURE(stid, grp1, grp2, dir, upctf);
   LOCAL newsid, %new stid%
      oldsid, %old stid being processed%
      newgp1; %head of new group%
   IF grp1.stfile # grp2.stfile THEN err($"illegal group");
   oldsid ← grp1;
   newsid ← newgp1 ← newrng(stid.stfile);
   LOOP
      BEGIN
      %get stid's, this branch%
      WHILE (oldsid := getsub(oldsid)) # oldsid DO
         newsid ← newsub(newsid);
      %now copy sdb's and point to next branch%
      LOOP
         BEGIN
         copsdb(oldsid, newsid);
         IF upctf THEN upctbl(oldsid, newsid, oldsid);
         IF oldsid = grp1 THEN %branch copy complete%
            BEGIN
            IF oldsid = grp2 THEN %group copy complete%
               BEGIN
               insgrp(stid, newgp1, newsid, dir);
               RETURN(newsid);
               END;
            oldsid ← grp1 ← getsuc(oldsid);
            EXIT;
```

```
            END;
         IF NOT getftl(oldsid := getsuc(oldsid)) THEN EXIT;
            %still more in plex%
         newsid ← getsuc(newsid);
         END;
      newsid ← newsuc(newsid);
      END;
   END.

(remgrp)
   %Removes group whose bounds are passed it from position in ring
   but does not delete either ring elements or SDB's.%
   %-------------%
   PROCEDURE(grp1, grp2);
   LOCAL stid; %work area for updating stid%
   IF grp1.stpsid = origin OR
      grp2.stpsid = origin THEN RETURN(FALSE);
   IF grp1.stfile # grp2.stfile THEN err($"illegal group");
   %putcdpanic(IF getfhd(grp1) THEN getup(grp1)
      ELSE getprd(grp1));%
   IF getfhd(grp1) THEN
      BEGIN
      stid ← getsuc(grp2); %new plex head%
      IF NOT getftl(grp2) THEN stofhd(stid, TRUE);
      stosub(getup(grp2), stid);
      END
   ELSE
      BEGIN
      stid ← getprd(grp1);
      stoftl(stid, getftl(grp2));
      stosuc(stid, getsuc(grp2));
      END;
   RETURN;
   END.

(delgrp)
   %This routine destroys all trace of the group bounded by the
   arguments passed it.  (Note that it does not remove the group from
   the ring; it assumes that this has been done by, for example,
   REMGRP.)
      It follows each branch to its deepest level, deleting
      references to substructure in the sub fields; it then follows
      the structure back up, through the suc pointers, freeing SDB's
      as it goes.%
   %-------------%
   PROCEDURE(grp1, grp2, newsid);
   LOCAL
      stid, %stid being processed%
      subsid; %sub of stid being processed%
   IF grp1.stfile # grp2.stfile THEN err($"illegal group");
   IF grp1.stpsid = origin OR
      grp2.stpsid = origin THEN err($"illegal delete");
   stid ← grp1;
   LOOP
      BEGIN
      WHILE (subsid ← getsub(stid)) # stid DO
```

```
            BEGIN
            stosub(stid, stid);
            stid ← subsid;
            END;
         stid ← getsuc(subsid); %now go back up%
         upctbl(subsid, endfil, newsid);
         <TXTEDT,dsttx> (subsid); %free sdb%
         frerng(subsid);
         IF subsid = grp2 THEN RETURN;
         END;
      END.

%move around in ring%

   (getail)
      %Given an stid, this procedure returns the stid of the tail of the
      current plex%
      %-------------%
      PROCEDURE (stid);
      IF stid.stpsid # origin THEN
         UNTIL getftl(stid) DO
         stid ← getsuc(stid);
      RETURN(stid);
      END.

   (getup)
      %Given an stid, this routine returns the stid of the source of the
      original stid%
      %-------------%
      PROCEDURE(stid);
      IF stid.stpsid # origin THEN
         stid ← getsuc(getail(stid));
      RETURN(stid);
      END.

   (gethed)
      %Given an stid, this routine returns the head of the current plex;
      if it is passed the origin statement, it returns the origin%
      %-------------%
      PROCEDURE(stid);
      IF stid.stpsid # origin THEN
         stid ← getsub(getup(stid));
      RETURN(stid);
      END.

   (getprd)
      %Given an stid, this routine returns the predecessor; if the psid
      heads a plex, the stid itself is returned%
      %-------------%
      PROCEDURE (stid);
      LOCAL presid, %stid of predecessor%
         sucsid; %stid of successor of presid%
      IF getfhd(stid) THEN RETURN(stid);
      presid ← gethed(stid);
      UNTIL (sucsid ← getsuc(presid)) = stid DO
         presid ← sucsid;
```

```
   RETURN(presid);
   END.

(getend)
   %This procedure returns the end of the tail of the stid passed
   it.%
   %-------------%
   PROCEDURE(stid);
   DO stid ← getail(stid)
   UNTIL (stid := getsub(stid)) = stid;
   RETURN(stid);
   END.

(getbck)
   %This procedure finds the back of the stid of the statemment
   passed it.  It does not observe viewspecs.%
   %-------------%
   PROCEDURE(stid);
   IF stid.stpsid = origin THEN RETURN(stid);
   IF getfhd(stid) THEN RETURN(getup(stid));
   stid ← getprd(stid);
   IF (stid := getsub(stid)) = stid THEN RETURN(stid);
   RETURN(getend(stid));
   END.

(getnxt)
   %This procedure finds the "next" statement, i.e. the sub,
   suc, father, grandfather, etc, of the STID passed as arg
   1.  It ignores all viewspec parameters.%
   %-----------------%
   PROCEDURE (stid);
   IF (stid := getsub(stid)) = stid THEN
      % no substructure %
      BEGIN
      LOOP
         BEGIN
         IF stid.stpsid = origin THEN RETURN (endfil);
         IF getftl(stid) = 0 THEN EXIT; % not a tail %
         stid ← getsuc(stid);
         END;
      stid ← getsuc(stid);
      END;
   IF stid.stpsid = origin THEN RETURN (endfil);
   RETURN (stid);
   END.


%correspondence table manipulation%

   (upctbl)
      %Given three stid's, this routine will update occurrences of the
      first stid, using rplsid as the stid that contains the text
      corresponding to oldsid, and newsid as the stid that is the
      replacement for oldsid.  (rplsid should be endfil if the original
      text has been eliminated.)%
      %-------------%
```

```
PROCEDURE(oldsid, rplsid, newsid);
LOCAL list, listnd;
REF list;
IF clstad = 0 OR [clstad].clbuff = 0 THEN RETURN;
&list ← [clstad].clbuff;
listnd ← &list + [clstad].clcnt * cll;
FOR &list UP cll UNTIL >= listnd
DO
   IF list.clst1 = oldsid THEN
      BEGIN
      list.clst2 ← newsid;
      list.clst1 ← rplsid;
      END
   ELSE
      IF list.clst2 = oldsid THEN
         list.clst2 ← newsid;
RETURN;
END.
```

FINISH of strmnp

SUBST

```
<NLS>SUBST.NLS;39, 10-NOV-71 9:42 MSC ;
FILE subst  %  L10  <REL-NLS>subst  %
%.....Fast substitute for NLS, 29-OCT-71 LPD.....%
%.....Documentation.....%
   %The test strings are sorted by initial character and chained
   together, with the head of the chain in a character-code indexed
   array subdsp.  This allows a very fast check whether a character can
   possibly begin a test string.  Each test-replacement pair is
   represented by a record (card) which is linked to others for the same
   initial test character through the canxt field.%
   %Global variables used:
      subcnt  count of substitutions
      lit     A-string for building up new statement
      swork   internal work area, see (stbpget)
   %
%.....Declarations.....%
   REGISTER a1=12, a2=13;
      %*** see main loop of substat ***%
%.....Common code for DNLS and TNLS.....%
   (substitute) PROCEDURE (type, stid1, stid2, sb, da);
   %substitute%
      %process the structure whose type and stid's are passed%
      %----------%
      subcnt ← 0;   % count of number of visibles %
      CASE type OF
         = stmtv: xsubs(stid1, sb, da);
         = brnchv: xsubb(stid1, sb, da);
         = plexv: xsubp(stid1, sb, da);
         = groupv: xsubg(stid1, stid2, sb, da);
         ENDCASE err(0);
      RETURN;
      END.

   (sbinit) PROCEDURE(sbhed, sbrec, sbdsp, sbastr);
      % initialize substitute data structure %
      LOCAL j;
      POINTER sbhed;
      sbhed.sbrp ← sbrec;
      sbhed.sbdp ← sbdsp;
      sbhed.sbas ← sbastr;
      *[sbastr]* ← NULL;
      FOR j ← 0 UP UNTIL >=200B DO [sbdsp+j] ← 0;
      RETURN END.

   (sbpush) PROCEDURE(str1, str2, hed);
      % add pair to substitute list %
      LOCAL astr, len, subrp, asa, cap;
      POINTER hed, subrp, cap;
      REF str1, str2, astr;
      &astr ← hed.sbas;
      len ← astr.L;
      asa ← &astr + 1; %origin of text for A-string%
      *astr* ← *astr*, *str1*, *str2*;
      subrp ← hed.sbrp;
      subrp.carnc ← str1.L;
      subrp.catnc ← str2.L;
```

```
      subrp.carbp ← conbp(asa,len);
      subrp.catbp ← conbp(asa,len+str1.L);
      subrp.canxt ← 0;
      %link card into appropriate chain%
         cap ← hed.sbdp + *str2*[1];
         IF [cap] = 0 THEN %empty chain%
            [cap] ← subrp
         ELSE BEGIN %link on end%
            cap ← [cap];
            WHILE cap.canxt DO cap ← cap.canxt;
            cap.canxt ← subrp;
         END;
      hed.sbrp ← subrp + lcard;
      RETURN END.

   DECLARE bpary = (010677777777B, 3507B8, 2607B8, 1707B8, 1007B8);
   (conbp) PROCEDURE(base,cn);
      % construct byte pointer %
      LOCAL q, r;
      DIV cn / 5, q, r;
      RETURN(base+bpary[r]+q);
      END.

%.....Core-NLS Substitute Code.....%

   (subbranch) PROCEDURE (stid, sb, da); %substitute branch%
      REF sb, da;
      xsubgrp (stid, stid, &sb, &da);
      RETURN;
      END.

   (subplex) PROCEDURE (stid, sb, da); %substitute plex%
      LOCAL stid1, stid2;
      REF sb, da;
      stid1 ← plxset (stid : stid2);
      xsubgrp (stid1, stid2, &sb, &da);
      RETURN;
      END.

   (subgroup) PROCEDURE (stid1, stid2, sb, da); %substitute group%
      REF sb, da;
      stid1 ← grptst (stid1, stid2 : stid2);
      xsubgrp (stid1, stid2, &sb, &da);
      RETURN;
      END.

   (xsubgrp) PROCEDURE    % runs substitution over a group %
      (stid1,   % stid of first branch in group %
      stid2,    % stid of last branch in group %
      sb,       % address of patterns %
      da);      % display area %
      LOCAL stid,
         sw;   % address of sequence work area %
      REF da, sb, sw;
      IF &da NOT IN [$dpyarea, $dpyarea + da1*dacnt) OR
         NOT da.daexis OR da.daempty THEN err(0);
```

```
   dismes(1, $"Substitute In Progress");
   &sw ← openseq (stid1, stid2, da.davspec, da.davspc2, da.dausqcod,
   da.dacacode);
   WHILE (stid ← seqgen (&sw)) # endfil DO substat(stid, &sb, &da);
   closeseq (&sw);
   dismes (0);
   RETURN;
   END.

(substat) PROCEDURE(stid, sb, da);
   % substitute in one statement %
   LOCAL
      cary,   %pointer (later SKIPN) to subdsp%
      sfc,    %first char position for scan%
      sbp,    %byte pointer to chars in statement%
      snc,    %counter for chars in statement%
      ch,     %last char read from statement%
      chbp,   %byte ptr to end of last change%
      chnc,   %char pos of last change from right end of original
      statement%
      chflag, %count of changes made%
      cap,    %ptr to substitution description record%
      cbp,    %byte ptr to candidate or replacement%
      cnc,    %length of candidate or replacement%
      cncl,   %length of candidate%
      tbp,    %temp sbp for comparison%
      wbp,    %byte ptr to last char written in new statement%
      wnc,    %number of chars written in new statement%
      maxsl;  %max size of new statement%
   POINTER sb, da, cap;
   cary ← sb.sbdp;
   sfc ← IF stid.stastr OR da.davspec.vsnamf THEN 1 ELSE
   fchtxt(stid);
   sbp ← stbpget(stid, sfc: snc);
   IF sfc = 1 THEN BEGIN %entire statement%
      chbp ← sbp; chnc ← snc;
   END ELSE %name not scanned%
      chbp ← stbpget(stid, 1: chnc);
   chflag ← 0;
   maxsl ← lit.M;
   wbp ← 0107B8 + $lit; %text begins at $lit+1%
   wnc ← 0;
   % set up for fast scan %
      a1 ← skipni;
      !HLLM a1,cary; %insert instruction part%
   (step1):
      BUMP snc;
   (step): %fast scan loop%
      !SOSG snc;
      !JRST done;
      !ILDB a1,sbp;
      !XCT cary; %SKIPN a2,subdsp(a1)%
      !JRST step;
      ch ← a1; cap ← a2;
      DO BEGIN
         IF (cnc ← (cncl ← cap.catnc)-1) < snc THEN BEGIN
```

```
            tbp ← sbp;
            cbp ← cap.catbp;
            ch ← ↑cbp; %skip first char%
            FOR cnc DOWN UNTIL <=0 DO
               IF ↑tbp # ↑cbp THEN GOTO noteq;
            % found match %
            IF (wnc ← wnc + chnc - snc + (cnc ← cap.carnc))
               > maxsl THEN BEGIN
                  % generate string system overflow error %
                  lit.L ← lit.M;
                  *lit* ← *lit*, SP;
                  RETURN; %if no trap - may not be possible%
               END;
            % copy segment between last match and this %
            sbp ← sbp + (IF sbp < 3507B8 THEN 07B10 ELSE
            437777777777B);
               %back up sbp by 1 char%
            UNTIL chbp = sbp DO
               ↑wbp ← ↑chbp;
            % enter replacement %
            cbp ← cap.carbp;
            FOR cnc DOWN UNTIL <=0 DO
               ↑wbp ← ↑cbp;
            % update variables %
            chbp ← sbp ← tbp;
            chnc ← snc ← snc - cncl;
            BUMP chflag;
            GOTO stepl;
         END;
         (noteq):
      END WHILE cap ← cap.canxt;
      GOTO step;
   (skipni): !SKIPN a2,0(a1); %instruction part for cary%
   (done):
   IF chflag THEN BEGIN
      IF (wnc ← wnc + chnc) > maxsl THEN abort();
      % copy rest of statement (past last match) %
      UNTIL chbp = sbp DO
         ↑wbp ← ↑chbp;
      lit.L ← wnc;
      dsttx(stid); %remove markers%
      ST stid ← *lit*;
      subcnt ← subcnt + chflag;
   END;
   RETURN END.

(stbpget) PROCEDURE (stid, cnt); %statement byte pointer get%
   LOCAL bp;
   swork ← stid; swork[1] ← cnt;
   fechcl(1, $swork);
   %swork[2] contains byte count+1, swork[4] has byte pointer%
   bp ← swork[4];
   IF bp .A 77B10 = 44B10 THEN
      bp ← bp - 430000000001B;
      %adjust char and word pos%
   RETURN(bp, swork[2]-1) END.
```

FINISH of subst

TCMNDS

```
<NLS>TCMNDS.NLS;25, 3-JAN-72 17:17 MSC ;
FILE tcmnds   %  L10   <REL-NLS>TCMNDS.REL  %

%.....Declarations.....%

   REF tda;
   REGISTER r1=1, r2=2, r3=3;

%.....Append........%

   (ta) PROCEDURE;        %APPEND STATEMENT COMMAND%
      echo($"Append ");
      ON SIGNAL
         = sighelp :
            BEGIN
            typeas($" (to) ADDR CA (from) ADDR CA LIT CA");
            RETURN;
            END;
         ELSE;
      tobl();
      LOOP
         BEGIN
         fromb2();
         *lit* ← NULL;
         txtlit($lit);
         mvcbl();
         xas($b1, $b2, $lit);
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN;
      END.

%.....Break........%

   (tb) PROCEDURE;
      echo($"Break ");
      echo($"Statement");
      ON SIGNAL
         = sighelp :
            BEGIN
            typeas($" (at) ADDR CA LEVADJ CA");
            RETURN;
            END;
         ELSE;
      LOOP
         BEGIN
         prompt($" at ");
         tbug($b1);
         levadj(b1, $ladj);
         CASE curchr OF
            = CA, =C. : NULL;
            ENDCASE error();
         *lit* ← NULL;
         b1 ← xbs($b1, $ladj, $lit);
         b1[1] ← 1;
```

```
      mvcbl();
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN;
   END.

%.....Copy........%

(tcmtl) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      = sighelp :
         BEGIN
         typeas($" (to) ADDR CA (from) ADDR CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      tobl();
      fromb2();
      mvcbl();
      proc($b1, $b2);
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tcc) PROCEDURE;
   echo($"Character");
   tcmtl($xcc);
   RETURN END.

(tcw) PROCEDURE;
   echo($"Word");
   tcmtl($xcw);
   RETURN END.

(tcn) PROCEDURE;
   echo($"Number");
   tcmtl($xcn);
   RETURN END.

(tcv) PROCEDURE;
   echo($"Visible");
   tcmtl($xcv);
   RETURN END.

(tci) PROCEDURE;
   echo($"Invisible");
   tcmtl($xci);
   RETURN END.

(tcl) PROCEDURE;
   echo($"Link");
```

```
   tcmt1($xc1);
   RETURN END.

(tcmt2) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      = sighelp :
         BEGIN
         typeas($" (to) ADDR CA (from) ADDR CA ADDR CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      tob1();
      fromb2();
      prompt($" ");
      tbug($b3);
      mvcb1();
      proc($b1, $b2, $b3);
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tct) PROCEDURE;
   echo($"Text");
   tcmt2($xct);
   RETURN END.

(tcms1) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      = sighelp :
         BEGIN
         typeas($" (to) ADDR CA (from) ADDR CA LEVADJ CA");
         RETURN;
         END;
      ELSE;
   tob1();
   LOOP
      BEGIN
      fromb2();
      levadj(b1, $ladj);
      CASE curchr OF
         = CA, =C. : NULL;
         ENDCASE error();
      b1 ← proc($b1, $b2, $ladj);
      b1[1] ← 1;
      mvcb1();
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tcs) PROCEDURE;
```

```
   echo($"Statement");
   tcms1($xcs);
   RETURN END.

(tcb) PROCEDURE;
   echo($"Branch");
   tcms1($xcb);
   RETURN END.

(tcp) PROCEDURE;
   echo($"Plex");
   tcms1($xcp);
   RETURN END.

(tcms2) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      = sighelp :
         BEGIN
         typeas($" (to) ADDR CA (from) ADDR CA ADDR CA LEVADJ CA");
         RETURN;
         END;
      ELSE;
   tobl();
   LOOP
      BEGIN
      fromb2();
      prompt($" ");
      tbug($b3);
      levadj(bl, $ladj);
      CASE curchr OF
         = CA, =C. : NULL;
         ENDCASE error();
      bl ← proc($bl, $b2, $b3, $ladj);
      bl[1] ← 1;
      mvcbl();
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tcg) PROCEDURE;
   echo($"Group");
   tcms2($xcg);
   RETURN END.

(tobl) PROCEDURE;
   prompt($" to ");
   tbug($bl);
   RETURN END.

(fromb2) PROCEDURE;
   prompt($" from ");
   tbug($b2);
   RETURN END.
```

```
%.....Delete........%

   (tdtl) PROCEDURE(proc);
      REF proc;
      ON SIGNAL
         = sighelp :
            BEGIN
            type$s($" (at) ADDR CA CA");
            RETURN;
            END;
         ELSE;
      LOOP
         BEGIN
         prompt($" at ");
         tbug($bl);
         prompt($" ok?");
         getctc();
         mvcbl();
         proc($bl);
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN END.

   (tdc) PROCEDURE;
      echo($"Character");
      tdtl($xdc);
      RETURN END.

   (tdw) PROCEDURE;
      echo($"Word");
      tdtl($xdw);
      RETURN END.

   (tdn) PROCEDURE;
      echo($"Number");
      tdtl($xdn);
      RETURN END.

   (tdv) PROCEDURE;
      echo($"Visible");
      tdtl($xdv);
      RETURN END.

   (tdi) PROCEDURE;
      echo($"Invisible");
      tdtl($xdi);
      RETURN END.

   (tdl) PROCEDURE;
      echo($"Link");
      tdtl($xdl);
      RETURN END.

   (tdt2) PROCEDURE(proc);
      REF proc;
```

```
   ON SIGNAL
      = sighelp :
         BEGIN
         typeas($" (at) ADDR CA ADDR CA CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      prompt($" at ");
      tbug($b1);
      prompt($" ");
      tbug($b2);
      prompt($" ok?");
      getctc();
      mvcbl();
      proc($b1, $b2);
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tdt) PROCEDURE;
   echo($"Text");
   tdt2($xdt);
   RETURN END.

(tds) PROCEDURE;
   echo($"Statement");
   tdt1($xds);
   RETURN END.

(tdb) PROCEDURE;
   echo($"Branch");
   tdt1($xdb);
   RETURN END.

(tdp) PROCEDURE;
   echo($"Plex");
   tdt1($xdp);
   RETURN END.

(tdg) PROCEDURE;
   echo($"Group");
   tdt2($xdg);
   RETURN END.

%.....Fix.......%

(tf) PROCEDURE;      %Fix Marker%
   ON SIGNAL
      = sighelp :
         BEGIN
         typeas($" LIT CA (at) ADDR CA");
         RETURN;
         END;
```

```
        ELSE;
      echo($"Fix marker");
      tfname();
      prompt($" at ");
      tbug($bl);
      mvcbl();
      xemf($bl, $lit);
      RETURN END.

   (tfname) PROCEDURE;
      prompt($" named ");
      echon();
      *lit* ← NULL;
      CASE lookc() OF % see if the first character is a ? %
         = SP:
            BEGIN
            input();
            REPEAT CASE;
            END;
         = '?:
            BEGIN
            input();
            SIGNAL(sighelp);
            END;
         ENDCASE;
      rdlit($lit, $rnmdel);
      RETURN END.

%.....Insert........%

   (titxt) PROCEDURE(proc);
      REF proc;
      ON SIGNAL
         = sighelp :
            BEGIN
            typeas($" (at) ADDR CA LIT CA");
            RETURN;
            END;
         ELSE;
      LOOP
         BEGIN
         prompt($" at ");
         tbug($bl);
         crlf();
         *lit* ← NULL;
         txtlit($lit);
         mvcbl();
         proc($bl, $lit);
         IF curchr # 0. THEN EXIT;
         crlf();
         END;
      RETURN END.

   (tic) PROCEDURE;
      echo($"Character");
      titxt($xic);
```

```
   RETURN END.

(tiw) PROCEDURE;
   echo($"Word");
   titxt($xiw);
   RETURN END.

(tin) PROCEDURE;
   echo($"Number");
   titxt($xin);
   RETURN END.

(tiv) PROCEDURE;
   echo($"Visible");
   titxt($xiv);
   RETURN END.

(tii) PROCEDURE;
   echo($"Invisible");
   titxt($xii);
   RETURN END.

(til) PROCEDURE;
   echo($"Link");
   titxt($xil);
   RETURN END.

(tit) PROCEDURE;
   echo($"Text");
   titxt($xic);
   RETURN END.

(tis) PROCEDURE;
   echo($"Statement");
   tinstructure();
   RETURN END.

(tinstructure) PROCEDURE;
   prompt($" at ");
   ON SIGNAL
      = sighelp :
         BEGIN
         typeas($" (at) ADDR CA LEVADJ LIT CA");
         RETURN;
         END;
      ELSE;
   tbug($bl);
   LOOP
      BEGIN
      levadj(bl, $ladj);
      *lit* ← NULL;
      txtlit($lit);
      bl ← xis($bl, $lit, $ladj);
      bl[1] ← 1;
      mvcbl();
      IF curchr # C. THEN RETURN;
```

```
      crlf();
      END;
   END.

%.....Move........%

(tmc) PROCEDURE;
   echo($"Character");
   tcmt1($xmc);
   RETURN END.

(tmw) PROCEDURE;
   echo($"Word");
   tcmt1($xmw);
   RETURN END.

(tmn) PROCEDURE;
   echo($"Number");
   tcmt1($xmn);
   RETURN END.

(tmv) PROCEDURE;
   echo($"Visible");
   tcmt1($xmv);
   RETURN END.

(tmi) PROCEDURE;
   echo($"Invisible");
   tcmt1($xmi);
   RETURN END.

(tml) PROCEDURE;
   echo($"Link");
   tcmt1($xml);
   RETURN END.

(tmt) PROCEDURE;
   echo($"Text");
   tcmt2($xmt);
   RETURN END.

(tms) PROCEDURE;
   echo($"Statement");
   tcms1($xms);
   RETURN END.

(tmb) PROCEDURE;
   echo($"Branch");
   tcms1($xmb);
   RETURN END.

(tmp) PROCEDURE;
   echo($"Plex");
   tcms1($xmp);
   RETURN END.
```

```
    (tmg) PROCEDURE;
       echo($"Group");
       tcms2($xmg);
       RETURN END.

%.....Name delimiters.........%

    (tns1) PROCEDURE(proc);
       LOCAL STRING str1[4], strr[4];
       LOCAL dlleft, dlrght;
       REF proc;
       ON SIGNAL
          = sighelp :
             BEGIN
             typeas($" (at) ADDR CA");
             RETURN;
             END;
          ELSE;
       LOOP
          BEGIN
          prompt($" at ");
          tbug($bl);
          crlf();
          prompt($"Left delimiter ");
          *str1* ← NULL;
          txtlit($str1);
          crlf();
          prompt($"Right delimiter ");
          *strr* ← NULL;
          txtlit($strr);
          prompt($" ok? ");
          getctc();
          todco(CA);
          dlleft ←
             IF str1.L =empty OR (FIND SF(*str1*) ("null"/"NULL")) THEN 0
             ELSE *str1*[1];
          dlrght ←
             IF strr.L =empty OR (FIND SF(*strr*) ("NULL"/"null")) THEN 0
             ELSE *strr*[1];
          proc($bl, dlleft, dlrght, tda);
          IF curchr # C. THEN EXIT;
          crlf();
          END;
       RETURN END.

    (tns2) PROCEDURE(proc);
       LOCAL STRING str1[4], strr[4];
       LOCAL dlleft, dlrght;
       REF proc;
       ON SIGNAL
          = sighelp :
             BEGIN
             typeas($" (at) ADDR CA");
             RETURN;
             END;
          ELSE;
```

```
   LOOP
      BEGIN
      prompt($" at ");
      tbug($b1);
      prompt($" ");
      tbug($b2);
      crlf();
      prompt($"Left delimiter ");
      *str1* ← NULL;
      txtlit ($str1);
      crlf();
      prompt($"Right delimiter ");
      *strr* ← NULL;
      txtlit ($strr);
      prompt($" ok? ");
      getctc();
      todco(CA);
      dlleft ←
         IF str1.L =empty OR (FIND SF(*str1*) ("null"/"NULL")) THEN 0
         ELSE *str1*[1];
      dlrght ←
         IF strr.L =empty OR (FIND SF(*strr*) ("NULL"/"null")) THEN 0
         ELSE *strr*[1];
      mvcbl();
      proc($b1, dlleft, dlrght, tda);
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tnd) PROCEDURE;
   LOCAL STRING str[15];
   echo($"Display");
   ON SIGNAL
      =sighelp:
         BEGIN
         typeas($" (at) ADDR CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      prompt($" at ");
      tbug($b1);
      mvcbl();
      <AUXCOD, getnmdls>(b1, $str);
      crlf();
      typeas($str);
      IF curchr # C. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tns) PROCEDURE;
   echo($"Statement");
   tnsl($xns);
```

```
      RETURN END.

   (tnb) PROCEDURE;
      echo($"Branch");
      tns1($xnb);
      RETURN END.

   (tnp) PROCEDURE;
      echo($"Plex");
      tns1($xnp);
      RETURN END.

   (tng) PROCEDURE;
      echo($"Group");
      tns2($xng);
      RETURN END.

%.....Print........%

   (tpr) PROCEDURE(type);
       ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA VIEWSPECS CA");
            RETURN;
            END;
         ELSE;
      tbug($bl);
      getvsp();
      treslev(bl);
      <TSPRT, printg>(bl, bl, type);
      RETURN;
      END.

   (tps) PROCEDURE;
      echo($"Statement ");
      tpr(stmtv);
      RETURN;
      END.

   (tpb) PROCEDURE;
      echo($"Branch ");
      tpr(brnchv);
      RETURN;
      END.

   (tpp) PROCEDURE;
      LOCAL stid1, stid2;
      echo($"Plex ");
      ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA VIEWSPECS CA");
            RETURN;
            END;
         ELSE;
```

```
      tbug($b1);
      getvsp();
      treslev(b1);
      stid1 ← plxset(b1 : stid2);
      <TSPT, printg>(stid1, stid2, plexv);
      RETURN;
      END.

   (tpg) PROCEDURE;
      ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA ADDR CA VIEWSPECS CA");
            RETURN;
            END;
         ELSE;
      echo($"Group ");
      tbug($b1);
      prompt($" ");
      tbug($b2);
      b1 ← <STRMNP, grptst>(b1, b2 : b2);
      getvsp();
      treslev(b1);
      <TSPRT, printg>(b1, b2, groupv);
      RETURN;
      END.

%.....Replace........%

   (trtl) PROCEDURE(proc);
      REF proc;
      ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA (by literal?) Yes LIT or No ADDR
            CA");
            GOTO STATE;
            END;
         ELSE;
      LOOP
         BEGIN
         treptext();
         mvcbl();
         proc(rplb, $b1, $b2, $lit);
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN END.

   (treptext) PROCEDURE;
      LOCAL char;
      prompt($" at ");
      tbug($b1);
      IF treptype() THEN
         BEGIN
         crlf();
```

```
      *lit* ← NULL;
      txtlit($lit);
      rplb ← FALSE;
      END
   ELSE
      BEGIN
      tbug($b2);
      rplb ← TRUE;
      END;
   RETURN END.

(trc) PROCEDURE;
   echo($"Character");
   trtl($xrc);
   RETURN END.

(trw) PROCEDURE;
   echo($"Word");
   trtl($xrw);
   RETURN END.

(trn) PROCEDURE;
   echo($"Number");
   trtl($xrn);
   RETURN END.

(trv) PROCEDURE;
   echo($"Visible");
   trtl($xrv);
   RETURN END.

(tri) PROCEDURE;
   echo($"Invisible");
   trtl($xri);
   RETURN END.

(trl) PROCEDURE;
   echo($"Link");
   trtl($xrl);
   RETURN END.

(trt2) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      =sighelp:
         BEGIN
         *lit* ← EOL,
            " (at) ADDR CA ADDR CA", EOL,
            " (by literal?) Yes LIT or No ADDR CA ADDR CA";
         typeas($lit);
         GOTO STATE;
         END;
      ELSE;
   LOOP
      BEGIN
      prompt($" at ");
```

```
      tbug($b1);
      prompt($" ");
      tbug($b2);
      IF treptype() THEN
         BEGIN
         crlf();
         *lit* ← NULL;
         txtlit($lit);
         rplb ← FALSE;
         END
      ELSE
         BEGIN
         tbug($b3);
         prompt($" ");
         tbug($b4);
         rplb ← TRUE;
         END;
      mvcbl();
      proc(rplb, $b1, $b2, $b3, $b4, $lit);
      IF curchr # C. THEN EXIT;
      END;
   RETURN END.

(trt) PROCEDURE;
   echo($"Text");
   trt2($xrt);
   RETURN END.

(treptype) PROCEDURE;
   prompt($" by literal? ");
   echoff();
   CASE inpcuc() OF
      =SP, =CA, ='Y:
         BEGIN
         prompt($" Yes");
         RETURN(TRUE);
         END;
      ='N:
         BEGIN
         prompt($" No");
         RETURN(FALSE);
         END;
      =CD: GOTO STATE;
      ENDCASE error();
   END.
(trsl) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      =signelp:
         BEGIN
         typeas($" (at) ADDR CA (by literal?) Yes LIT or No ADDR
         CA");
         GOTO STATE;
         END;
      ELSE;
   treptext();
```

```
   bl ← proc(rplb, $bl, $b2, $lit);
   bl[1] ← 1;
   mvcbl();
   trepstructure();
   RETURN END.

(trs) PROCEDURE;
   echo($"Statement");
   trsl($xrs);
   RETURN END.

(trb) PROCEDURE;
   echo($"Branch");
   trsl($xrb);
   RETURN END.

(trp) PROCEDURE;
   echo($"Plex");
   trsl($xrp);
   RETURN END.

(trg) PROCEDURE;
   LOCAL char;
   ON SIGNAL
      =sighelp:
         BEGIN
         *lit* ← EOL,
            " (at) ADDR CA ADDR CA", EOL,
            " (by text?) Yes TEXT or No ADDR CA ADDR CA";
         typeas($lit);
         GOTO STATE;
         END;
      ELSE;
   echo($"Group");
   prompt($" at ");
   tbug($bl);
   prompt($" ");
   tbug($b2);
   IF treptype() THEN
      BEGIN
      crlf();
      *lit* ← NULL;
      txtlit($lit);
      rplb ← FALSE;
      END
   ELSE
      BEGIN
      tbug($b3);
      prompt($" ");
      tbug($b4);
      rplb ← TRUE;
      END;
   bl ← xrg(rplb, $bl, $b2, $b3, $b4, $lit);
   bl[1] ← 1;
   mvcbl();
   trepstructure();
```

```
    RETURN END.

(trepstructure) PROCEDURE;
    WHILE curchr = C. DO
        BEGIN
        crlf();
        levadj(bl, $ladj);
        *lit* ← NULL;
        txtlit($lit);
        bl ← xis($bl, $lit, $ladj);
        bl[1] ← 1;
        mvcbl();
        END;
    RETURN END.

%.....Substitute........%

(tssl) PROCEDURE(type);
    ON SIGNAL
        =sighelp:
            BEGIN
            *lit* ← EOL,
                " (at) ADDR CA", EOL,
                " (literal) LIT CA (for literal) LIT CA (go?)", EOL,
                " Yes or No (literal) ... ";
            typeas($lit);
            GOTO STATE;
            END;
        ELSE;
    LOOP
        BEGIN
        prompt($" at ");
        tbug($bl);
        tsubcn(type, bl);
        bl[1] ← 1;
        mvcbl();
        IF curchr # C. THEN EXIT;
        crlf();
        END;
    RETURN END.

(tsubs) PROCEDURE;
    echo($"Statement");
    tssl(stmtv);
    RETURN END.

(tsubb) PROCEDURE;
    echo($"Branch");
    tssl(brnchv);
    RETURN END.

(tsubp) PROCEDURE;
    LOCAL stid1, stid2;
    ON SIGNAL
        =sighelp:
            BEGIN
```

```
            *lit* ← EOL,
               " (at) ADDR CA", EOL,
               " (literal) LIT CA (for literal) LIT CA (go?) Yes or No";
            typeas($lit);
            GOTO STATE;
            END;
         ELSE;
      echo($"Plex");
      LOOP
         BEGIN
         prompt($" at ");
         tbug($bl);
         stid1 ← <STRMNP, plxset> (bl : stid2);
         tsubcn(plexv, stid1, stid2);
         bl[1] ← 1;
         mvcbl();
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN END.

(tsubg) PROCEDURE;
      LOCAL stid1, stid2;
      ON SIGNAL
         =sighelp:
            BEGIN
            *lit* ← EOL,
               " (at) ADDR CA ADDR CA", EOL,
               " (literal) LIT CA (for literal) LIT CA (go?) Yes or No";
            typeas($lit);
            GOTO STATE;
            END;
         ELSE;
      echo($"Group");
      LOOP
         BEGIN
         prompt($" at ");
         tbug($bl);
         prompt($" ");
         tbug($b2);
         stid1 ← <STRMNP, grptst> (bl, b2 : stid2);
         tsubcn(groupv, stid1, stid2);
         bl[1] ← 1;
         mvcbl();
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN END.

(tsubcn) PROCEDURE(type, stid1, stid2);
      %-----------------%
      LOCAL count;
      LOCAL STRING rstr[80], tstr[80];
      LOCAL subdsp[200B];
      *auxlit* ← NULL;
      count ← 0;
```

```
      CALL sbinit($subhed, $subrec, $subdsp, $auxlit);
      LOOP  %collect patterns%
         BEGIN
         BUMP count;
         crlf();
         *rstr* ← NULL;
         prompt($" Literal ");
         txtlit($rstr); %get pattern%
         *tstr* ← NULL;
         prompt($" For ");
         txtlit($tstr);
         <SUBST, sbpush>($rstr, $tstr, $subhed); %push strings%
         IF count = $subnum THEN EXIT;
         prompt($" Go? ");
         IF answer() THEN EXIT;
         END;
      crlf();
      <SUBST, substitute>(type, stid1, stid2, $subhed, &tda);
      crlf();
      typeas($"substitutions = ");
      typeno(subcnt, 0); %type number of substitutions%
      RETURN;
      END.

%.....Transpose........%

   (ttt1) PROCEDURE(proc);
      REF proc;
      ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA (and) ADDR CA");
            RETURN;
            END;
         ELSE;
      LOOP
         BEGIN
         prompt($" at ");
         tbug($b1);
         prompt($" and ");
         tbug($b2);
         mvcb1();
         proc($b1, $b2);
         IF curchr # 0. THEN EXIT;
         crlf();
         END;
      RETURN END.
   (ttc) PROCEDURE;
      echo($"Character");
      ttt1($xtc);
      RETURN END.

   (ttw) PROCEDURE;
      echo($"Word");
      ttt1($xtw);
      RETURN END.
```

```
(ttn) PROCEDURE;
   echo($"Number");
   ttt1($xtn);
   RETURN END.

(ttv) PROCEDURE;
   echo($"Visible");
   ttt1($xtv);
   RETURN END.

(tti) PROCEDURE;
   echo($"Invisible");
   ttt1($xti);
   RETURN END.

(ttl) PROCEDURE;
   echo($"Link");
   ttt1($xtl);
   RETURN END.

(ttt2) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      =sighelp:
         BEGIN
         typeas($" (at) ADDR CA ADDR CA (and) ADDR CA ADDR CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      prompt($" at ");
      tbug($b1);
      prompt($" ");
      tbug($b2);
      prompt($" and ");
      tbug($b3);
      prompt($" ");
      tbug($b4);
      mvcbl();
      proc($b1, $b2, $b3, $b4);
      IF curchr # 0. THEN EXIT;
      crlf();
      END;
   RETURN END.
(ttt) PROCEDURE;
   echo($"Text");
   ttt2($xtt);
   RETURN END.

(tts) PROCEDURE;
   echo($"Statement");
   ttt1($xts);
   RETURN END.
```

```
    (ttb) PROCEDURE;
       echo($"Branch");
       ttt1($xtb);
       RETURN END.

    (ttp) PROCEDURE;
       echo($"Plex");
       ttt1($xtp);
       RETURN END.

    (ttg) PROCEDURE;
       echo($"Group");
       ttt2($xtg);
       RETURN END.

%.....view specs........%

    (tv) PROCEDURE;
       echo($"View specs: ");
       ON SIGNAL
          =sighelp:
             BEGIN
             *lit* ← EOL,
                " a   (L ← L-1)", EOL,
                " b   (L ← L+1)", EOL,
                " c   (L ← ALL)", EOL,
                " d   (L ← 1)", EOL,
                " e   (L ← REL)", EOL,
                " g   (branch only on)", EOL,
                " h   (branch only off)", EOL,
                " i   (content analyzer on)", EOL;
             *lit* ← *lit*,
                " j   (content analyzer off)", EOL,
                " k   (content analyzer fail)", EOL,
                " l   (plex only)", EOL,
                " m   (statement numbers on)", EOL,
                " n   (statement numbers off)", EOL,
                " q   (T ← T-1)", EOL,
                " r   (T ← T+1)", EOL,
                " s   (T ← ALL)", EOL,
                " w   (L ← T ← ALL)", EOL;
             *lit* ← *lit*,
                " x   (L ← T ← 1)", EOL,
                " y   (blank lines on)", EOL,
                " z   (blank lines off)", EOL,
                " A   (indenting on)", EOL,
                " B   (indenting off)", EOL,
                " C   (names on)", EOL,
                " D   (names off)", EOL,
                " K   (signiture on)", EOL,
                " L   (signiture off)";
             typeas($lit);
             RETURN;
             END;
          ELSE;
       getvsp();
```

```
      treslev(tda.dacsp); %resolve relative level wrt dacsp%
      RETURN;
      END.

%.....Xset........%

   (txtl) PROCEDURE(proc);
      REF proc;
      ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA");
            RETURN;
            END;
         ELSE;
      LOOP
         BEGIN
         prompt($" at ");
         tbug($bl);
         mvcbl();
         proc($bl);
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN END.
   (txc) PROCEDURE;
      echo($"Character");
      txtl($xxc);
      RETURN END.

   (txw) PROCEDURE;
      echo($"Word");
      txtl($xxw);
      RETURN END.

   (txn) PROCEDURE;
      echo($"Number");
      txtl($xxn);
      RETURN END.

   (txv) PROCEDURE;
      echo($"Visible");
      txtl($xxv);
      RETURN END.

   (txi) PROCEDURE;
      echo($"Invisible");
      txtl($xxi);
      RETURN END.

   (txl) PROCEDURE;
      echo($"Link");
      txtl($xxl);
      RETURN END.

   (txt2) PROCEDURE(proc);
```

```
   REF proc;
   ON SIGNAL
      =sighelp:
         BEGIN
         typeas($" (at) ADDR CA ADDR CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      prompt($" at ");
      tbug($b1);
      prompt($" ");
      tbug($b2);
      mvcb1();
      proc($b1, $b2);
      IF curchr # 0. THEN EXIT;
      crlf();
      END;
   RETURN END.
(txt) PROCEDURE;
   echo($"Text");
   txt2($xxt);
   RETURN END.

(txs) PROCEDURE;
   echo($"Statement");
   txtl($xxs);
   RETURN END.

(txb) PROCEDURE;
   echo($"Branch");
   txtl($xxb);
   RETURN END.

(txg) PROCEDURE;
   echo($"Group");
   txt2($xxg);
   RETURN END.

(txp) PROCEDURE;
   echo($"Plex");
   txtl($xxp);
   RETURN END.

(txm) PROCEDURE;
   CASE inpcuc() OF
      = 'C:
         BEGIN
         echo($"Capital ");
         xsucm();
         REPEAT;
         END;
      = 'L:
         BEGIN
         echo($"Lower ");
```

```
            xslcm();
            REPEAT;
            END;
         = '?:
            BEGIN
            *lit* ←
               EOL,
               "Capital ", EOL,
               "Lower ";
            typeas($lit);
            RETURN;
            END;
         = SP: REPEAT CASE;
         = CD: GOTO STATE;
         = CA:
            BEGIN
            todco(CA);
            RETURN;
            END;
      ENDCASE error();
   END.

FINISH
```

TCIL

```
<NLS>TCTL.NLS;94, 2-JAN-72 22:05 BLP ;
FILE tctl   %  L10   <REL-NLS>TCTL.REL  %

REF tda;

%.....Primary Command Language Routine.....%

   (mainct)PROCEDURE; %command language for todas%
      %-----------------%
      LOCAL count; %for feedback line indenting%
      ON SIGNAL =-4: %Called from rubout% reset(); ELSE SIGNAL;
      <INPFBK, croot>();
      LOOP
         BEGIN
         crlf();
         count ← 0;
         WHILE (count ← count + 1) <= fedind DO typech(SP);
         typech('*);
         echoff();
         input();
         ttywc();
         END;
      END.

   (ttywc)
      %This routine determines the command being specified and transfers
      control to the appropriate routine.%
      PROCEDURE;
      LOCAL char, newversion;
      IF (char ← curchr) IN ['a, 'z] THEN char ← char - 40B;
      CASE char OF
         = 'A:  %append%
            <TCMNDS, ta> ();
         = 'B:  %break%
            <TCMNDS, tb> ();
         = 'C:  %copy%
            BEGIN
            echo($"Copy ");
            CASE inpcuc() OF
               = 'C: tcc(); %copy character%
               = 'W: tcw(); %copy word%
               = 'N: tcn(); %copy number%
               = 'V: tcv(); %copy visible%
               = 'I: tci(); %copy invisible%
               = 'L: tcl(); %copy link%
               = 'T: tct(); %copy text%
               = 'S: tcs(); %copy statement%
               = 'B: tcb(); %copy branch%
               = 'P: tcp(); %copy plex%
               = 'G: tcg(); %copy group%
               = '?: tyhelp(0); %type entities for edit commands%
               = SP: REPEAT CASE;
               ENDCASE error();
            END;
         = 'D:              %Delete Command%
            BEGIN
```

```
        echo($"Delete ");
        CASE inpcuc() OF
            = 'C: tdc(); %delete character%
            = 'W: tdw(); %delete word%
            = 'N: tdn(); %delete number%
            = 'V: tdv(); %delete visible%
            = 'I: tdi(); %delete invisible%
            = 'L: tdl(); %delete link%
            = 'T: tdt(); %delete text%
            = 'S: tds(); %delete statement%
            = 'B: tdb(); %delete branch%
            = 'P: tdp(); %delete plex%
            = 'G: tdg(); %delete group%
            = '?: tyhelp(0); %type entities for edit commands%
            = SP: REPEAT CASE;
           ENDCASE error();
        END;
    = 'E: <TXCT, texecute>(); %Execute Command%
    = 'F: <TCMNDS, tf>();          %Fix Marker%
    = 'G: <TXCT, tgoto>();      %Goto comand%
    = 'I:                      %Insert Command%
        BEGIN
        echo($"Insert ");
        CASE inpcuc() OF
            = 'C: tic(); %insert character%
            = 'W: tiw(); %insert word%
            = 'N: tin(); %insert number%
            = 'V: tiv(); %insert visible%
            = 'I: tii(); %insert invisible%
            = 'L: til(); %insert link%
            = 'T: tit(); %insert text%
            = 'S,
            = 'B,
            = 'P,
            = 'G: tis(); %insert statement%
            = '?:
               BEGIN
               *lit* ← EOL, "Statement", EOL;
               tyhp(2);
               typeas($lit);
               END;
            = SP: REPEAT CASE;
           ENDCASE error();
        END;
    = 'L: <IOCTL, tl> ();              %Load Command%
    = 'M:                    %Move Command%
        BEGIN
        echo($"Move ");
        CASE inpcuc() OF
            = 'C: tmc(); %move character%
            = 'W: tmw(); %move word%
            = 'N: tmn(); %move number%
            = 'V: tmv(); %move visible%
            = 'I: tmi(); %move invisible%
            = 'L: tml(); %move link%
            = 'T: tmt(); %move text%
```

```
        = 'S: tms(); %move statement%
        = 'B: tmb(); %move branch%
        = 'P: tmp(); %move plex%
        = 'G: tmg(); %move group%
        = '?: tyhelp(0); %type entities for edit commands%
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= 'N:               %Name delimiter Command%
    BEGIN
    echo($"Name delimiter ");
    CASE inpcuc() OF
        = 'D: tnd(); %Name delimiter display%
        = 'S: tns(); %Name delimiter statement%
        = 'B: tnb(); %Name delimiter branch%
        = 'P: tnp(); %Name delimiter plex%
        = 'G: tng(); %Name delimiter group%
        = '?:
            BEGIN
            *lit* ←
                EOL,
                "Display ";
            tyhp(1);
            typeas($lit);
            END;
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= '0: <IOCTL, to> ();                %Output Command%
= 'P:                       %Print Command%
    BEGIN
    echo($"Print ");
    CASE inpcuc() OF
        = 'S: tps(); %print statement%
        = 'B: tpb(); %print branch%
        = 'P: tpp(); %print plex%
        = 'G: tpg(); %print group%
        = '?: tyhelp(1); %type structural entities%
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= 'R:                       %Replace Command%
    BEGIN
    echo($"Replace ");
    CASE inpcuc() OF
        = 'C: trc(); %replace character%
        = 'W: trw(); %replace word%
        = 'N: trn(); %replace number%
        = 'V: trv(); %replace visible%
        = 'I: tri(); %replace invisible%
        = 'L: trl(); %replace link%
        = 'T: trt(); %replace text%
        = 'S: trs(); %replace statement%
        = 'B: trb(); %replace branch%
        = 'P: trp(); %replace plex%
        = 'G: trg(); %replace group%
```

```
        = '?: tyhelp(0); %type entities for edit commands%
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= 'S: %<TSPRT,sbstut>(); Substitute Command%
    BEGIN
    echo($"Substitute ");
    CASE inpcuc() OF
        = 'S: tsubs(); %substitute statement%
        = 'B: tsubb(); %substitute branch%
        = 'P: tsubp(); %substitute plex%
        = 'G: tsubg(); %substitute group%
        = '?: tyhelp(1); %type structural entities%
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= 'T:                   %Transpose Command%
    BEGIN
    echo($"Transpose ");
    CASE inpcuc() OF
        = 'C: ttc(); %transpose character%
        = 'W: ttw(); %transpose word%
        = 'N: ttn(); %transpose number%
        = 'V: ttv(); %transpose visible%
        = 'I: tti(); %transpose invisible%
        = 'L: ttl(); %transpose link%
        = 'T: ttt(); %transpose text%
        = 'S: tts(); %transpose statement%
        = 'B: ttb(); %transpose branch%
        = 'P: ttp(); %transpose plex%
        = 'G: ttg(); %transpose group%
        = '?: tyhelp(0); %type entities for edit commands%
        = SP: REPEAT CASE;
        ENDCASE error();
    END;
= 'U: <IOCTL, tuf>();    %Update File%
= 'V: <TCMNDS,tv>();     %View specs Command%
= 'X:                    %Xset Command%
    BEGIN
    echo($"Xset ");
    CASE inpcuc() OF
        = 'C: txc(); %set character%
        = 'W: txw(); %set word%
        = 'N: txn(); %set number%
        = 'V: txv(); %set visible%
        = 'I: txi(); %set invisible%
        = 'L: txl(); %set link%
        = 'T: txt(); %set text%
        = 'S: txs(); %set statement%
        = 'B: txb(); %set branch%
        = 'P: txp(); %set plex%
        = 'G: txg(); %set group%
        = 'M: %set mode%
            BEGIN
            echo($"Mode ");
            txm();
```

```
                END;
             = '?:
                BEGIN
                *lit* ←
                   EOL,
                   "Mode ";
                typeas($lit);
                tyhelp(0);
                END;
             = SP: REPEAT CASE;
             ENDCASE error();
          END;
       = '.: tpoint();          %Show point%
       = ';: tcmnt();           %Comment Command%
       = '/:                    %Slash Command%
          BEGIN
          todco('/);
          tslash(tda.dacsp, tda.daccnt);
          END;
       = '\: tbslash();         %Backslash Command%
       = '↑: tua();             %Up Arrow Command%
       = LF: tlinfed();         %Line Feed Command%
       = SP: taddr();           %Get new address%
       = $ascalt: taltm();      %Repeat Search%
       =CD, =CA, =EOL, =CR, =C.: NULL;
       =todlit:
          REPEAT CASE (char ← rawchr());
       = '?: tyhelp(-1);        %Help%
       ENDCASE error();
    RETURN;
    END.

FINISH of tctl
```

TSPRT

<NLS>TSPRT.NLS;30, 14-MAR-72 22:33 BLP ;

```
FILE tsprt  %  L10  <REL-NLS>TSPRT  %

%.....Declarations.................%

   REGISTER r1 = 1;
   REF tda;
   DECLARE STRING dashes = "----------";

   %.....Global Declarations.........%

      % GLOBAL typnwa, typend, feedbk, fedind, caflg, sublnk, buffsz,
      buffct, buff, curchr, todlit, trnsli, trnlso, pshift, tshift,
      cshft0, wshft0, pshft0, tablst, subcnt, textex, textfg, rdtext,
      stn2, swork, tprtpt;%

%.....Print Routines..............%

   (printg) PROCEDURE (stid1, stid2, type);
      %print the structure (type) addrssed by stid1,stid2%
      %------------------%
      LOCAL
         char,   %current character%
         gapcol, %column of character preceding last invisible char%
         gapptr, %pointer to character preceding last invisible char%
         startp, %pointer to start of line character%
         prindt,  %number of columns to indent%
         lincnt, %count of lines printed for current statement%
         stid,   %current stid being printed%
         maxcol, %maximum number of columns for current line%
         stnrt,  %whether statement numbers are to be printed on right%
         sw,     %address of sequence generator work area%
         head,   %address of file header%
         mkrptr; %marker table pointer%
         LOCAL STRING
            str[100],    %scratch string%
            stnsig[50];  %holds statement number or signature%
         REF sw, mkrptr;

      IF stid1 = endfil THEN error();
      lineno ← IF type = stmtv THEN 0    %do not eject page%
         ELSE tda.damrow / tda.davinc;   %set to eject page%
      &sw ←
         <SEQGEN, openseq> (stid1, stid2, tda.davspec, tda.davspc2,
         tda.dausqcod, tda.dacacode);
      bl[0] ← IF stid1.stastr THEN tda.dacsp ELSE stid1;
      UNTIL ((stid ← seqgen (&sw)) = endfil) OR
        (type = stmtv AND sw.swcstid # stid1) DO
         BEGIN
         %set up work area s2work for READC%
            s2work ← stid;
            s2work[1] ←
               IF tda.davspec.vsnamf OR stid.stastr THEN 1 %print name%
               ELSE fchtxt (getsdb (stid)); %skip past name%
            fechcl (forward, $s2work);
```

```
%markers (not allowed for now)%
   prmkrf ← 0;
   IF FALSE AND tda.davspec.vsmkrf AND NOT stid.stastr THEN
      BEGIN  %see if marker in statemnt%
      &mkrptr ← $mkrtb - $filhed +
         (head ← <FILMNP, filhdr>(stid.stfile));
      mkrend ← &mkrptr +
         (($mkrtbl - $filhed + head) * mkrl;
      mkrflg ← FALSE;
      FOR mkrptr UP mkrl UNTIL = mkrend DO
         IF mkrptr.mkpsid = stid.stpsid THEN
            BEGIN
            mkrflg ← TRUE;
            mkrcnt ← mkrptr.mkccnt + 1;
            EXIT;
            END;
      END;
prindt ←        %indentation%
   IF tda.davspec.vsindf THEN
      MAX (0, MIN (tda.daind * (sw.swclvl-1), tda.damind,
      spst.M))
   ELSE 0;
*prbuf* ← *spst* (empty + 1 TO prindt);
maxcol  ← tda.damcol;   %max no. cols%
stnrt   ← FALSE;
IF tda.davspec.vsstnf AND stid.stpsid # origin AND
 NOT stid.stastr THEN
   IF NOT tda.davspec.vsstnr THEN
      BEGIN  %print statement number%
      fechnm (sw.swsvw, $prbuf);
      *prbuf* ← *prbuf*, SP;
      END
   ELSE   % statement numbers go on the right %
      BEGIN
      *stnsig* ← NULL;
      fechnm (sw.swsvw, $stnsig);
      maxcol ← maxcol - stnsig.L - 1;
      stnrt  ← TRUE;
      END;
%line printing loop%
FOR lincnt ← 0 UP UNTIL = tda.davspec.vstrnc DO
   BEGIN
   IF inptrf THEN EXIT 2;
   gapcol ← tda.daccol ← prbuf.L * tda.dahinc;
   gapptr ← startp ← prbuf.L;
   UNTIL tda.daccol >= maxcol DO
      CASE char ← READC ($s2work) OF
         =SP:
            BEGIN
            gapptr     ← prbuf.L;
            gapcol     ← tda.daccol;
            *prbuf*    ← *prbuf*, SP;
            tda.daccol ← tda.daccol + 1;
            END;
         =ENDCHR, =EOL, =CR:
            BEGIN
```

tda.daccol ← tda.daccol + putchr(char, tda.daccol, &prbuf);

```
                gapptr      ← prbuf.L;
                gapcol      ← tda.daccol;
                EXIT LOOP;
                END;
             =TAB:
                BEGIN
                gapptr      ← prbuf.L;
                gapcol      ← tda.daccol;
                *prbuf*     ← *prbuf*, TAB;
                tda.daccol ← fndtab (&tda, tda.daccol + 1);
                END;
             ENDCASE
                BEGIN
                *prbuf* ← *prbuf*, char;
                tda.daccol ← tda.daccol + 1;
                END;
       %by here, a line has been constructed. fix gapptr and
       gapcol if there were no invisibles%
          IF startp = gapptr THEN   %no in invisibles in line%
             BEGIN
             gapptr ← prbuf.L;
             gapcol ← tda.daccol;
             END;
       IF stnrt THEN
          BEGIN
          stnrt    ← FALSE;
          maxcol   ← tda.damcol;   %max no. cols%
          *str*     ← *prbuf* [gapptr + 2 TO prbuf.L];
          *prbuf* [gapptr + 1 TO prbuf.L] ←
             *spst* [gapcol + 1 TO maxcol - stnsig.L],
             *stnsig*;
          IF NOT prtype (prbuf.L) THEN EXIT 2; %type the line%
          *prbuf* [prindt + 1 TO prbuf.L] ← *str*;
          END
       ELSE
          BEGIN
          IF NOT prtype (gapptr) THEN EXIT 2; %type the line%
          *prbuf* [prindt + 1 TO prbuf.L] ←
             *prbuf* [gapptr + 2 TO prbuf.L];
          END;
       bl[0] ← stid;
       IF char = ENDCHR THEN EXIT;
       END;
    %blank line and signature%
       IF tda.davspec.vsidtf AND NOT stid.stastr THEN
          BEGIN
          *stnsig* ← NULL;
          fechsig (stid, $stnsig);
          *prbuf* ←
             *spst* [1 TO maxcol - stnsig.L],
             *stnsig*;
          prtype (prbuf.L);
          END
       ELSE IF tda.davspec.vsblkf THEN crlf();
    END;
 <SEQGEN, closeseq> (&sw);
```

```
      crlf ();
      bl[1] ← 1;
      mvcbl(); %Move control marker to bl%
      RETURN;
      END.

   (prtype) PROCEDURE (number);
      %type 'number' characters from prbuf%
      %------------------%
      LOCAL length; %length of astring in prbuff%
      %wait for output buffer empty%
         rl ← 101B; !JSYS dobe;
      IF inptrf %↑O% THEN RETURN (FALSE);
      IF inpstp %↑S% THEN
         BEGIN
         inpstp ← FALSE;
         RETURN (FALSE);
         END;
      IF tda.davspec.vspagf AND tda.dacrow >= tda.damrow THEN newpag()
      ELSE crlf();
      tda.dacrow ← tda.dacrow + tda.davinc;
      length ← prbuf.L := number;
      typeas ($prbuf);
      prbuf.L ← length;
      RETURN (TRUE);
      END.

   (newpag)  %page eject%
      % this procedure does a page eject for printing in todas. It
      takes into account the device being used, and tries to make the
      total page length equal to 11 1/2 inches, unless otherwise
      specified by the user%
      %------------------%
      PROCEDURE;
      LOCAL top, bottom, extra;
      DIV (tda.dabottom-tda.datop-tda.damrow)/2, top, extra;
      bottom ← top + extra;
         %put extra line (if it exists) at the bottom%
      crlf();
      DO crlf() UNTIL (bottom ← bottom - tda.davinc) <=
      tda.davinc;
      IF bottom > 0 THEN
         BEGIN
         typeas($dashes);
         crlf();
         END;
      UNTIL (top := top - tda.davinc) <= 0 DO crlf();
      tda.dacrow ← 0;
      RETURN;
      END.

%.....Verify command code.........%

   (todvfy) PROCEDURE; %todas entry to File Verify%
      %------------------%
      LOCAL vrfy;
```

```
    LOOP
        BEGIN
        echo($"Verify ");
        IF answer() THEN
            BEGIN
            vrfy ← TRUE;
            EXIT;
            END;
        crlf();
        prompt($"Checksum ");
        IF answer() THEN
            BEGIN
            vrfy ← FALSE;
            EXIT;
            END;
        crlf()
        END;
    crlf();
    <VERIFY, vfmain>(tda.dacsp.stfile, vrfy);
    GOTO STATE;
    END.

%.....secondary i/o routines......%

    (answer) %this procedure accepts a yes or no answer from the
    keyboard, and returns with a 0 if the answer was negative, and a 1
    if it was positive%
        %------------------%
        PROCEDURE;
        echoff();
        CASE inpcuc() OF
            = 'Y, = CA:
                BEGIN
                echo($"Yes");
                RETURN(TRUE);
                END;
            = 'N, = SP:
                BEGIN
                echo($"No");
                RETURN(FALSE)
                END;
            = CD: GOTO STATE;
            ENDCASE
                BEGIN
                typeas($" ?");
                REPEAT;
                END;
        END.

    (gttnum) PROCEDURE (oldvlu);
        %read a number for TODAS from tty, accepting the old value if the
        user types a command accept as the first character%
        %------------------%
        LOCAL value;
        echon();
        CASE lookc() OF
```

```
      = CA: RETURN(oldvlu);
      NOT IN ['0, '9]: error();
      ENDCASE
         BEGIN
         value ← 0;
         WHILE lookc() IN ['0, '9] DO
            value ← value * 10 + (input() - '0);
         RETURN(value);
         END;
   END.

(typeno) PROCEDURE (number, astrng);  %type number on tty%
   %type the number which is passed on the tty if astrng is FALSE;
   otherwise put number in astrng.  Build the characters in the work
   area typnwa.  This routine can handle negative numbers also%
   %------------------%
   LOCAL worka, quotnt, remain, sign;
   REF astrng, worka;
   quotnt ← number;
   &worka ← $typnwa;
   DO
      BEGIN
      DIV quotnt/10, quotnt, remain;
      worka ← remain + '0;
      END
   UNTIL ((&worka ← &worka + 1) = $typend) OR (quotnt = 0);
   IF number < 0 THEN &worka ← '-
   ELSE BUMP DOWN &worka;
   DO
      IF &astrng THEN *astrng* ← *astrng*, worka
      ELSE
         BEGIN
         r1 ← worka;
         !JSYS pbout;
         END
   UNTIL (&worka ← &worka -1) < $typnwa;
   RETURN;
   END.

FINISH
```

TXCMND

```
<NLS>TXCMND.NLS;50, 3-JAN-72 16:44 MSC ; ["linkfg"];
FILE txcmnd   % L10   <REL-NLS>TXCMND.REL %

%.....Declarations.....%

   REF tda;
   REGISTER r1=1, r2=2, r3=3;

%.....Misc commands........%

   (tshptr) PROCEDURE(stid, cnt, astrng);
      REF astrng;
      *astrng* ← *astrng*, " = ";
      fechno(stid, &astrng);
      *astrng* ← *astrng*, " +", STRING(cnt);
      RETURN;
      END.

   (tpoint) PROCEDURE;
      todco('.);
      *stn* ← NULL;
      tshptr(tda.dacsp, tda.daccnt, $stn);
      typeas($stn);
      RETURN;
      END.

   (taltm) PROCEDURE;
      LOCAL ldel, rdel;
      LOCAL STRING treps[50];
      bl ← tda.dacsp;
      bl[1] ← tda.daccnt;
      CASE srchtype OF
         = word:
            BEGIN
            ldel ← '<;
            rdel ← '>;
            END;
         = contnt:
            BEGIN
            ldel ← '[;
            rdel ← ']';
            END;
         = contls:
            ldel ← rdel ← ';;
         ENDCASE error();
      *treps* ← SP, ldel, *conreg*, rdel;
      typeas($treps);
      typech(CA);
      specreg($conreg, srchtype, $bl);
      IF bl = endfil THEN error();
      mvcbl();
      RETURN;
      END.

   (tlinfed) PROCEDURE;
      LOCAL sw, stid;
```

```
   REF sw;
   &sw ←
      <SEQGEN, openseq>(getnxt(tda.dacsp), endfil, tda.davspec,
      tda.davspc2, tda.dausqcod, tda.dacacode);
   IF seqgen(&sw) = endfil THEN error();
   stid ← sw.swcstid;
   <SEQGEN, closeseq>(&sw);
   <TSPRT, printg>(stid, stid, stmtv);
   RETURN;
   END.

(taddr) PROCEDURE;
   todco(SP);
   ON SIGNAL
      = sighelp:
         BEGIN
         typeas($" ADDR CA");
         RETURN;
         END;
      ELSE;
   tbug($bl);
   mvcbl();
   RETURN;
   END.

(tua) PROCEDURE;
   LOCAL stid;
   todco('↑);
   IF (stid ← getbck(tda.dacsp)) = endfil THEN error();
   <TSPRT, printg>(stid, stid, stmtv);
   RETURN;
   END.

(tbslash) PROCEDURE;
   todco('\);
   tbsprnt(tda.dacsp);
   RETURN;
   END.

(tbsprnt) PROCEDURE(stid);
   % print the statement save and restore current CM %
   <TSPRT, movpr>(stid, stid, stmtv);
   RETURN;
   END.

(tslash) PROCEDURE(stid, cnt);
   LOCAL TEXT POINTER tcm, ptr;
   tcm ← ptr ← stid;
   tcm[1] ← cnt;
   ptr[1] ← MAX(1, tcm[1]-tslshchars);
   *lit* ← SP, ptr tcm, '<, LF, '>;
   FIND SE(tcm) ↑ptr;
   ptr[1] ← MIN(tcm[1]+tslshchars+1, ptr[1]);
   *lit* ← *lit*, tcm ptr;
   typeas($lit);
   RETURN;
```

```
    END.

(tcmnt) PROCEDURE;
    LOCAL char;
    todco('");
    WHILE (char ← input()) # CA DO todco(char);
        %echo comment characters%
    todco('");
    RETURN;
    END.

%.....Initialization and Error Routines.......%

(tstart)PROCEDURE;
    %-----------------%
    reset();
    END.

(reset) PROCEDURE;
    %-----------------%
    zap();
    STATE ← tstate, spec;
    mainct(); %start command parsing again%
    END.

(error) PROCEDURE;
    %-----------------%
    textfg ← FALSE;
    typech('?);
    clrbuf(0);  %clear todas input buffers %
    GOTO STATE;
    END.

%.....Primary Command Language I/O Routines...%

(tbug) PROCEDURE(ptr);
    REF ptr;
    ptr ← tda.dacsp;
    ptr[1] ← tda.daccnt;
    getadr(&ptr);
    RETURN END.

(getadr) PROCEDURE(ptr); %read statement address from tty%
    %parameter: address of tpointer=starting tpointer for
    relative addresses
    Returns: after having updated the tpointer
    Error conditions: calls error if receives unusual input
    %
    %----------------%
    REF ptr;
    LOCAL TEXT POINTER oldptr;
    LOCAL STRING
        gder[50],  %for error string%
        gadstr[100]; % to hold the entire string for the address %
    (gad):
    IF tadlfg THEN linkfg ← FALSE;
```

```
   *gadstr* ← NULL;
   gadlit($gadstr);%collect the literal%
   oldptr ← ptr;
   oldptr[1] ← ptr[1];
   ON SIGNAL %catch any errors during interpretation%
      =gaderr:
         BEGIN
         *[MESSAGE]*←SP,*[MESSAGE]*,"?  ";
         typeas(MESSAGE);
         clrbuf(0);
         ptr ← oldptr;
         ptr[1] ← oldptr[1];
         GOTO gad;
         END;
      ELSE;
   gadgo($gadstr, $gder, &ptr, &tda); %interpret the string%
   tadlfg ← FALSE;
   RETURN END.

(gadlit) PROCEDURE(astrng); %read statement address from tty%
   %read a literal from the keyboard for getadr%
   %-----------------%
   LOCAL char, stid, litescape;
   REF astrng;
   echon();
   litescape ← todlit;
      %keep in local environment to avoid page fault to read on each
      character processed%
   CASE lookc() OF % see if the first character is a ? %
      = SP:
         BEGIN
         input();
         REPEAT CASE;
         END;
      = '?:
         BEGIN
         input();
         SIGNAL(sighelp);
         END;
      ENDCASE;
   LOOP CASE char ← input() OF
      = CD:  %command delete%
         GOTO STATE;
      = BC:  %backspace character%
         IF astrng.L > empty THEN
            BEGIN
            IF (char ← *astrng*[astrng.L]) = $ascalt THEN char ← '$;
            todco(char);
            bkc(&astrng);
            END;
      = BW:  %backspace word%
         BEGIN
         todco(BW);
         CASE *astrng*[astrng.L] OF
            = SP, = TAB, = CR:
               %space past invisible characters%
```

```
            IF astrng.L > empty THEN
               CASE bkc(&astrng) OF
                  = SP, = TAB, = CR: REPEAT 1;
                  ENDCASE;
         ENDCASE;
      LOOP CASE bkc(&astrng) OF
         =SP, =TAB, =CR, =ENDCHR: EXIT LOOP;
         ENDCASE;
      END;
   = $ascbst: %backspace statement%
      BEGIN
      *astrng* ← NULL;
      crlf();
      END;
   = $ctlr:   %retype literal%
      BEGIN
      crlf();
      typeas(&astrng);
      END;
   IN [SP,'←], IN ['a,'z], =TAB,=CR,=EOL,=LF,=litescape,=$ascalt:
      BEGIN
      CASE char OF
         =litescape: %literal escape character%
            BEGIN
            *astrng* ← *astrng*, char;
               %gadgo needs the escape char%
            char ← rawchr();
            END;
         =$ascalt: %altmode%
            todco('$);
         ENDCASE;
      *astrng* ← *astrng*, char;
      END;
   =CA, =C.: RETURN;
   ENDCASE
      dismes(2, $"*Illegal Character*");
END.

(gadgo) PROCEDURE(gadstr, gder, ptr, da);
   % called with a string that defines an address and a starting
   location %
   LOCAL char, count, cnt, oldcnt, scnbck;
   LOCAL STRING ggst[50];
   LOCAL TEXT POINTER pscan;
   REF gadstr, gder, ptr, da;
   count ← 1;
   cnt ← empty;
   LOOP
      BEGIN
      *gder* ← char ← *gadstr*[cnt ← cnt+1];
      CASE char OF
         =SP, =todlit:  % space means nothing %
            NULL;
         ='?: SIGNAL(sighelp);
         ='/:   % slash %
            IF nlmode # fulldisplay THEN tslash(ptr, ptr[1]);
```

```
='\:   % backslash %
   IF nlmode # fulldisplay THEN tbsprnt(ptr);
='S, ='s:  % succ %
   getpr($getsuc, count, &ptr);
='P, ='p:  % predecessor %
   getpr($getprd, count, &ptr);
='D, ='d:  % down %
   getpr($getsub, count, &ptr);
='U, ='u:  % up %
   getpr($getup, count, &ptr);
='H, ='h:  % head %
   getpr($gethed, count, &ptr);
='T, ='t:  % tail %
   getpr($getail, count, &ptr);
='E, ='e:  % end %
   BEGIN
   ptr ← <NLS, JUMP, getvnd>(ptr, da.davspec.vslev);
   ptr[1] ← 1;
   END;
='N, ='n:  % next %
   getpr($getnxt, count, &ptr);
='B, ='b:  % back %
   getpr($getbck, count, &ptr);
='R, ='r:  % return %
   IF count < 0 THEN
      BEGIN
      count ← -count;
      REPEAT('A);
      END
   ELSE
      BEGIN
      ptr ← getret(&da, count);
      ptr[1] ← 1;
      END;
='A, ='a:  % ahead %
   IF count < 0 THEN
      BEGIN
      count ← -count;
      REPEAT('R);
      END
   ELSE
      BEGIN
      ptr ← getahd(&da, count);
      ptr[1] ← 1;
      END;
='&: % Jump file return %
   gadjf(&ptr, $popls, &gder, count, &da);
='@: % Jump file ahead %
   gadjf(&ptr, $bumpls, &gder, count, &da);
='↑: % indirect  link %
   BEGIN
   oldcnt ← cnt;
   ptr[1] ← MAX(<NLS, FILMNP, fchtxt>(getsdb(ptr)), ptr[1]);
   <JUMP, lnkspec>(count, &ptr,
      $stno, $stn2, $stn, $num);
   IF stn2.L # empty THEN ptr ← <JUMP, nfstid> ($stn2, $stn)
```

```
        ELSE ptr ← <JUMP, ofstid> (&da, $stn);
        ptr[1] ← 1;
        IF num.L # empty THEN feedlt(&da, $num);
        IF tadlfg THEN linkfg ← -1;
        *gder* ← *gadstr*[oldcnt TO cnt];
        END;
    ='(:  % link %
        BEGIN
        oldcnt ← cnt;
        *ggst* ← '(;
        cnt ← gdlit2(&gadstr, cnt+1, $ggst, '));
        *ggst* ← *ggst*, ');
        FIND SF(*ggst*) ↑pscan;
        <JUMP, lnkspec>(1, $pscan,
           $stno, $stn2, $stn, $num);
        IF stn2.L # empty THEN ptr ← <JUMP, nfstid> ($stn2, $stn)
        ELSE ptr ← <JUMP, ofstid> (&da, $stn);
        ptr[1] ← count ← 1;
        IF num.L # empty THEN feedlt(&da, $num);
        IF tadlfg THEN linkfg ← -1;
        *gder* ← *gadstr*[oldcnt TO cnt];
        END;
    ='.:  % name %
        BEGIN
        oldcnt ← cnt;
        cnt ← gadname(&ptr, &gadstr, cnt+1, $ggst);
        *gder* ← *gadstr*[oldcnt TO cnt];
        END;
    ='::  % error %
        err($"Use .name instead of :name");
    =$ascalt: % repeat last search %
        BEGIN
        oldcnt ← cnt;
        cnt ← gadfspec(&ptr, srchtype, &gadstr, cnt);
        UNTIL (count ← count-1) < 0 DO
           specreg($conreg, srchtype, &ptr);
        *gder* ← '$, *gadstr*[oldcnt+1 TO cnt];
        END;
    ='[: %content search%
        cnt ←
        conspt(&ptr,&gadstr,cnt+1,count,$ggst,&gder,contnt,']);
    ='<: %word search%
        cnt ←
        conspt(&ptr,&gadstr,cnt+1,count,$ggst,&gder,word,'>);
    ='; %semicolon% : %content search in statement%
        cnt ←
        conspt(&ptr,&gadstr,cnt+1,count,$ggst,&gder,contls,';);
    ='' %apostrophe% : %character search%
        BEGIN
        oldcnt ← cnt;
        IF *gadstr* [cnt ← cnt+1] = todlit THEN cnt ← cnt+1;
        *ggst* ← *gadstr* [cnt];
        *gder* ← *gadstr* [oldcnt TO cnt];
        specreg ($ggst, contnt, &ptr);
        IF ptr # endfil THEN
           BEGIN
```

UNTIL (count ← count-1) < 0 DO ✓

```
        *conreg* ← *ggst*;
        srchtype ← contnt;
        END;
    END;
='#: % marker %
    BEGIN
    oldcnt ← cnt := gdlit2(&gadstr, cnt+1, $ggst, SP);
    ptr ← <INPFBK, lkmkr>($ggst, da.dacsp.stfile : ptr[1]);
    *gder* ← *gadstr*[oldcnt TO cnt];
    END;
='+: %scan right%
    BEGIN
    count ← gadnum(&gadstr, cnt+1 : cnt);
    CASE *gadstr*[cnt+1] OF
        =SP, =ENDCHR: REPEAT 2 ('C);
        ENDCASE REPEAT LOOP;
    END;
='-: %scan left%
    BEGIN
    count ← -gadnum(&gadstr, cnt+1 : cnt);
    CASE *gadstr*[cnt+1] OF
        =SP, =ENDCHR: REPEAT 2 ('C);
        ENDCASE REPEAT LOOP;
    END;
IN ['0, '9]:
    BEGIN
    count ← gadnum(&gadstr, cnt : cnt);
    CASE *gadstr*[cnt+1] OF
        =SP, =ENDCHR: REPEAT 2 ('C);
        ENDCASE REPEAT LOOP;
    END;
='W, ='w: %word scan%
    BEGIN
    count ← gaddir(&ptr, count : scnbck);
    UNTIL (count ← count-1) < 0 DO
        FIND $LD $NLD;
    IF scnbck THEN FIND $LD;
    FIND ↑ptr;
    END;
='I, ='i: %invisible scan%
    BEGIN
    count ← gaddir(&ptr, count : scnbck);
    UNTIL (count ← count-1) < 0 DO
        FIND $NP $PT;
    IF scnbck THEN FIND $NP;
    FIND ↑ptr;
    END;
='V, ='v: %visible scan%
    BEGIN
    count ← gaddir(&ptr, count : scnbck);
    UNTIL (count ← count-1) < 0 DO
        FIND $PT $NP;
    IF scnbck THEN FIND $PT;
    FIND ↑ptr;
    END;
='C, ='c: %character scan%
```

```
            BEGIN
            count ← gaddir(&ptr, count);
            UNTIL (count ← count-1) < 0 DO FIND CH;
            FIND ↑ptr;
            END;
         ='←: %statement front%
            FIND SF(ptr) ↑ptr;
         ='>: %statement end%
            FIND SE(ptr) ↑ptr;
         =ENDCHR: %have finished interpreting the string%
            RETURN;
         ENDCASE
            SIGNAL(gaderr, &gder);
      count ← 1;
      IF ptr = endfil THEN SIGNAL(gaderr, &gder);
      IF POS ptr = SE(ptr) THEN FIND ptr < CH ↑ptr;
      END;  % of loop %
   END.
(conspt) PROCEDURE(ptr,astr1,cnt,count,astr2,astr3,type,stopchar);
   % used for content and word searches.  uses gadspt to extract and
   do the first execution. then repeats as many times as indicated
   by count. returns new value for cnt. %
   LOCAL oldcnt;
   REF astr1, astr3;
   oldcnt ← cnt := gadspt(ptr, &astr1, cnt, astr2, type, stopchar);
   UNTIL (count ← count-1) = 0 DO
      specreg(astr2, type, ptr);
   *astr3* ← *astr1*[oldcnt-1 TO cnt];
   RETURN(cnt) END.
(gadfspec) PROCEDURE(ptr, type, gadstr, cnt);
   % looks for F specification on searches %
   REF ptr, gadstr;
   CASE *gadstr*[cnt+1] OF
      =SP, =todlit:
         BEGIN
         BUMP cnt;
         REPEAT CASE;
         END;
      ='f, ='F:
         BEGIN
         BUMP cnt;
         IF type # contls THEN
            ptr.stpsid ← origin;
         ptr[1] ← empty;
         END;
      ENDCASE;
   RETURN(cnt) END.

(gadjf) PROCEDURE(ptr, proc, gder, count, da);
   LOCAL linkp, fileno, linkpf;
   REF ptr, proc, da;
   IF count < 0 THEN
      BEGIN
      &proc ← IF &proc = $popls THEN $bumpls ELSE $popls;
      count ← -count;
      END;
```

```
   linkpf ← da.dalink;
   UNTIL (count ← count-1) < 0 DO
      IF NOT(linkp ← da.dalink ← proc(&da)) THEN
         BEGIN
         da.dalink ← linkpf;
         SIGNAL(gaderr, gder);
         END;
   fileno ← jmpopn(/linkp/.lfilnm);
   restjs(linkp, fileno);
   ptr ← readjs(&da);
   ptr[1] ← 1;
   IF tadlfg THEN linkfg ← linkp;
   da.dalink ← linkpf;
   RETURN;
   END.

(gadspt) PROCEDURE(ptr, astr1, cnt, astr2, type, stopchar);
   LOCAL char;
   REF ptr, astr1, astr2;
   *astr2* ← NULL;
   % collect the string %
   cnt ← gdlit2(&astr1, cnt, &astr2, stopchar);
   % look for F specification %
   cnt ← gadfspec(&ptr, type, &astr1, cnt);
   specreg(&astr2, type, &ptr);
   IF ptr # endfil AND type # name THEN
      BEGIN
      srchtype ← type;
      *conreg* ← *astr2*;
      END;
   RETURN (cnt) END.

(gadname) PROCEDURE(ptr, astr1, cnt, astr2);
   LOCAL TEXT POINTER tp1, tp2, tp3;
   REF ptr, astr1, astr2;
   tp1 ← <UTILTY, asrref>(&astr1);
   tp1[1] ← cnt;
   <TXTEDT, nmdr>($tp1, $tp2, $tp3);
   *astr2* ← tp2 tp3;
   % look for F specification %
   cnt ← gadfspec(&ptr, name, &astr1, tp3[1/-1);
   specreg(&astr2, name, &ptr);
   RETURN(cnt) END.

(gaddir) PROCEDURE(ptr,count);
   REF ptr;
   IF count < 0 THEN
      BEGIN
      FIND ptr < ;
      RETURN ( -count, TRUE );
      END;
   FIND ptr > ;
   RETURN ( count, FALSE ) END.
(getpr) PROCEDURE(proc, count, ptr);
   REF proc, ptr;
   IF count < 0 THEN %do the inverse%
```

```
      BEGIN
      &proc ← CASE &proc OF
         = $getsuc : $getprd;
         = $getprd : $getsuc;
         = $getsub : $getup;
         = $getup  : $getsub;
         = $gethed : $getail;
         = $getail : $gethed;
         = $getnxt : $getbck;
         = $getbck : $getnxt;
         ENDCASE &proc;
      count ← -count;
      END;
   UNTIL (count ← count-1) < 0 DO
      BEGIN
      IF &proc = $getsuc AND getftl(ptr) THEN EXIT;
         % thus successor of statement with no successor is NOP %
      ptr ← proc(ptr);
      ptr[1] ← 1;
      END;
   RETURN END.
(gdlit2) PROCEDURE(astl, cnt, ast2, stopchar);
   % append characters to ast2 from astl starting at cnt until
   encounter a stopchar or the end of astl.  Return the count in astl
   of the stopchar. %
   LOCAL char;
   REF astl, ast2;
   LOOP
      BEGIN
      char ← *astl*[cnt];
      CASE char OF
         =ENDCHR, =stopchar:
            RETURN(cnt);
         =todlit:
            char ← *astl*[cnt ← cnt+1];
         ENDCASE;
      *ast2* ← *ast2*, char;
      BUMP cnt;
      END;
   END.
(gadnum) PROCEDURE(astr, cnt);
   % extract and evaluate number from string for gadgo.  Called with
   address of a string and a count specifying where to start looking
   for a number in the string.  Returns (value, cnt) where cnt gives
   the count which when incremented gives the character after the
   number. %
   LOCAL count, char;
   REF astr;
   IF (char ← *astr*[cnt]) NOT IN ['0, '9] THEN RETURN(1, cnt-1);
   count ← char - '0;
   WHILE (char ← *astr*[cnt ← cnt+1]) IN ['0, '9] DO
      count ← count*10 + char - '0;
   RETURN(count, cnt-1) END.

(tnumber)PROCEDURE(number);
   %Read a number for TNLS%
```

```
   %Return lentgh of number read%
   REF number;
   *number* ← NULL;
   WHILE input() IN ['0, '9] DO
      BEGIN
      *number* ← *number*, curchr;
      todco(curchr);
      END;
   RETURN(number.L);
   END.
(nqmlit) PROCEDURE(astrng, chrlst);
   % If next character is a '? then SIGNAL(sighelp,0) else equivalent
   to rdlit. %
   %------------------%
   IF lookc() = '? THEN
      BEGIN
      input();
      SIGNAL(sighelp,0);
      END;
   RETURN(rdlit(astrng, chrlst)) END.
(rdlit) PROCEDURE (astrng, chrlst);  %read literal from keyboard%
   %read a literal from the keyboard, doing all of the control
   things, plus breaking on any control character identified by the
   list in chrlst% %returns 0 if buffer overflows, 1 if normal
   termination (ca or c.), 2 if a break character was input.%
   %------------------%
   LOCAL char, stid;
   REF astrng;
   %assumes echon%
   LOOP
      BEGIN
      char ← input();
      IF chrlst > 0 AND chrpos(char, chrlst) THEN
         RETURN(2, char);
            %break character%
      CASE char OF
         = todlit:  %literal escape%
            BEGIN
            char ← rawchr();
            *astrng* ← *astrng*, char;
            END;
         = CD:  %command delete%
            GOTO STATE;
         = CA, = C.:  %command accept or center dot%
            RETURN(1, char);
         = BC:  %backspace character%
            IF astrng.L > empty THEN
               BEGIN
               todco(*astrng*[astrng.L]);
               bkc(&astrng);
               END;
         = BW:  %backspace word%
            BEGIN
            todco(BW);
            CASE *astrng*[astrng.L] OF
               = SP, = TAB, = CR:
```

```
                %space past invisible characters%
                IF astrng.L > empty THEN
                   CASE bkc(&astrng) OF
                      = SP, = TAB, = CR: REPEAT 1;
                      ENDCASE;
                ENDCASE;
             LOOP CASE bkc(&astrng) OF
                =SP, =TAB, =CR, =ENDCHR: EXIT LOOP;
                ENDCASE;
             END;
          = $ascbst: %backspace statement%
             BEGIN
             *astrng* ← NULL;
             todco(EOL);
             END;
          = $ctlr:   %retype literal%
             BEGIN
             crlf();
             typeas(&astrng)
             END;
          IN [SP, '←], IN ['a, 'z], =TAB, =CR, =EOL, =LF, =ALT:
             BEGIN
             IF astrng.L = astrng.M THEN
                RETURN(0, char);
                   %astrng full--return error condition%
             *astrng* ← *astrng*, char;
             END;
          ENDCASE dismes(2, $"*Illegal Character*");
       END;
    END.

(txtlit) PROCEDURE(astrng); %read a literal into A-string%
    % This procedure reads a literal into the A-string whose
    address is passed.  It also checks for control-y (enter
    alte mode) and returns alterv as the completion code. %
    %-----------------%
    LOCAL rcndtn;
    echon(); %**for now call echon (later echolt)**turn off
    break on every character%
    IF (rcndtn ← rdlit(astrng, 0)) = 0 THEN err(6);
    echoff(); % turn break back on, echo off%
    RETURN (rcndtn);
    END.

(getctc) PROCEDURE;
    %scan for a ctc, and returns if ok. Otherwise goes to error%
    %-----------------%
    CASE input() OF
       =CA, =C., ='Y, ='y: RETURN;
       ENDCASE error();
    END.

(getvsp)  %get view specs from user%
    % this procedure accepts a string of viewspecs from the
    keyboard and then calls feedlt to put them into effect.  After
    calling this procedure, should call tresleV(stid) to have relative
```

```
   level made absolute wrt level of stid %
   %-----------------%
   PROCEDURE;
   *num* ← NULL;
   echon();
   LOOP
      CASE lookc() OF
         =SP:
            BEGIN
            input();
            REPEAT CASE;
            END;
         ='?:
            BEGIN
            input();
            SIGNAL(sighelp);
            END;
         ENDCASE EXIT;
   IF rdlit($num, 0) = 0 THEN err(6);
   IF num.L > empty THEN <JUMP, feedlt>(&tda, $num);
   RETURN;
   END.

(vspstatus)PROC(da, astr, actflag);
   %Puts status of current viewspecs in english into astring
   according to flag: flag = false, all viewspecs. flag = true, only
   viewspec which effect staement selection and are not standard%
   LOCAL vsp;
   REF astr, da;
   *astr* ← NULL;
   IF actflag THEN
      BEGIN
      END
   ELSE
      BEGIN
      END;
   RETURN(FALSE);
   END.

(treslev) PROCEDURE(stid);
   IF tda.davspec.vsrlev THEN
      tda.davspec ← <SEQGEN, reslev>(tda.davspec, getlev(stid));
   RETURN END.

(levadj) PROCEDURE (stid, astrng);
   % this procedure conducts dialogue with the user to do level
   adjusting (returns after updating astring), given an
   initial stid and string address. %
   %-----------------%
   LOCAL char;
   REF astrng;
   echoff();
   todco(SP);
   IF lvdjnm THEN
      BEGIN
      *stno* ← NULL;
```

```
        IF stid.stpsid = origin THEN fechnd(stid, $stno)
           ELSE fechnp(stid, $stno);
        typeas($stno);
        typech(SP);
        END;
     *astrng* ← NULL;
     LOOP
        BEGIN
        CASE (char ← lookc()) OF
           =CA, =C., =SP:
              BEGIN
              todco(input());
              EXIT LOOP;
              END;
           ='U, ='u, ='D, ='d, IN ['0, '9]:
              BEGIN
              *astrng* ← *astrng*, char;
              todco(char);
              END;
           ='?:
              BEGIN
              input();
              SIGNAL(sighelp);
              END;
           =BC:
              IF astrng.L > empty THEN
                 BEGIN
                 char ← *astrng*[astrng.L];
                 todco(BC);
                 todco(char);
                 astrng.L ← astrng.L - 1;
                 END;
           =BW, =$ascbst:
              BEGIN
              todco(char);
              *stn* ← NULL;
              END;
           =CD: GOTO STATE;
           ENDCASE EXIT LOOP;
           input();
        END;
     crlf();
     CASE char OF
        =CA, =C., =SP: NULL;
        ENDCASE todco(char);
     echon();
     RETURN;
     END.

%.....Support Routines.......%

   (clrbuf) %clear the todas input and (optionally) output
   buffers%
      %if outflg is non-zero the output buffer is also cleared.
      nothing is  returned%
      %-----------------%
```

```
   PROCEDURE (outflg);
      IF outflg THEN
         BEGIN
         %jsys to clear output buffer%
            r1 ← 101B;
            !JSYS cfobf;
         END;
      %jsys to clear input buffer%
         r1 ← 100B;
         !JSYS cfibf;
      buffct ← 0; %clear the todas buffer%
      RETURN;
      END.

(chrpos) PROCEDURE (char, astrng); %scan string for character%
   %chr in first parameter, a-string addr in second%
   %return TRUE if char in string, FALSE otherwise.%
   %-----------------%
   LOCAL tmpchr;
   REF astrng;
   CCPOS *astrng*;
   LOOP
      IF (tmpchr ← READC) = char THEN RETURN(TRUE)
      ELSE IF tmpchr = ENDCHR THEN RETURN(FALSE);
   END.

(astruc) PROCEDURE(astrng); %a-string to upper case%
   % convert a-string (in astrng) to upper case %
   %-----------------%
   LOCAL length, bytptr, char;
   REF astrng;
   IF (length ← astrng.L) = empty THEN RETURN(&astrng);
   bytptr ← chbptr(empty) + &astrng;
   DO IF (char ← ↑bytptr) IN ['a,'z] THEN
      .bytptr ← char - 40B
   UNTIL (length ← length - 1) = empty;
   RETURN(&astrng);
   END.

(mvcbl) PROCEDURE;
   mvcsp($bl);
   RETURN END.

(mvcsp) PROCEDURE(ptr);
   LOCAL fl;
   REF ptr;
   IF linkfg = -1 THEN %link return or ahead followed by link%
      BEGIN
      pushls(&tda, [fl←flntadr(ptr.stfile)].flastr);
      tda.dacsp ← ptr;
      tda.daempty ← FALSE;
      pushjs(&tda);
      mesfre(fl, ptr);
      END
   ELSE
      BEGIN
```

```
        IF linkfg#0 THEN %link return or ahead, no other links%
           BEGIN
           tda.dalink ← linkfg;
           setdpa(&tda, linkfg);
           fl ← flntadr(ptr.stfile);
           mesfre(fl, ptr);
           END;
        tda.dacsp ← ptr;
        IF readjs(&tda)#tda.dacsp THEN pushjs(&tda);
        END;
     tda.daccnt ← ptr[1];
     tadlfg ← TRUE;
     linkfg ← FALSE;
     RETURN;
     END.

  (mesfre)PROCEDURE(fl, ptr);
     REF fl;
     dismes(2, fl.flastr);
     IF fl.fllock THEN lockmes(ptr.stfile);
     freflnt();
     RETURN;
     END.

%.....Help.....%

  (tyhelp) PROCEDURE(type);
     CASE type OF
        =-1:  %type list of all commands%
           BEGIN
           *lit* ←
              EOL,
              "Append ", EOL,
              "Break ", EOL,
              "Copy ", EOL,
              "Delete ", EOL,
              "Execute ", EOL,
              "Fix ", EOL,
              "Goto ", EOL,
              "Insert ", EOL;
           *lit* ← *lit*,
              "Load ", EOL,
              "Move ", EOL,
              "Name delimiters ", EOL,
              "Output ", EOL,
              "Print ", EOL,
              "Replace ", EOL,
              "Substitute ", EOL,
              "Transpose ", EOL,
              "Update ", EOL,
              "Viewspecs ", EOL;
           *lit* ← *lit*,
              "Xset ", EOL,
              ". (show location)", EOL,
              "/ (print context)", EOL,
              "\ (print statement)", EOL,
```

ELSE upjscnt (tda

```
                "↑ (move back and print)", EOL,
                "LF (line feed, move forward and print)", EOL,
                "SP ADDR CA (space, move to new address)", EOL,
                "ALT (altmode, repeat last search)";
             END;
          = 0:  %type the entities for edit commands%
             BEGIN
             *lit* ← EOL;
             tyhp(1);
             *lit* ← *lit*, EOL;
             tyhp(2);
             END;
          = 1, =2:  %type the structural or textual entities%
             BEGIN
             *lit* ← EOL;
             tyhp(type);
             END;
          ENDCASE;
       typeas($lit);
       RETURN END.

    (tyhp) PROCEDURE(type);
       CASE type OF
          = 1:  %structural entities%
             *lit* ← *lit*,
                "Statement ", EOL,
                "Branch ", EOL,
                "Plex ", EOL,
                "Group ";
          = 2:  %textual entities%
             *lit* ← *lit*,
                "Character ", EOL,
                "Word ", EOL,
                "Number ", EOL,
                "Visible ", EOL,
                "Invisible ", EOL,
                "Link ", EOL,
                "Text ";
          ENDCASE;
       RETURN END.

FINISH of txcmnd
```

TXCT

```
<NIC-NLS>TXCT.NLS;8, 11-JAN-72 22:23 BLP ;
FILE txct  % L10  <NIC-NLS>TXCT.REL %

%.....Declarations.................%

   SET %for opcodes%
      LSH=242B, SETCM=460B;
   REGISTER r1=1, r2=2, r3=3, r4=4;
   REF tda;
   DECLARE forward=1, uselit=0;

%.....TNLS Execute commands parser........%

   (texecute) PROCEDURE;
      %------------------%
      LOCAL fileno, strtyp, jfn, filtyp, flags, errs, savsp1, savsp2;
      LOCAL TEXT POINTER stid, stid1, stid2;
      LOCAL STRING  str[400], flnam[100], levj[20];
      echo ($"Execute ");
      CASE inpcuc() OF
         = '?:
            help(
                 $"Assimilate ",
                 $"Browse ",
                 $"Catalog Numbers ",
                 $"Content Analyzer ",
                 $"Device Specification",
                 $"Edit ",
                 $"File Verify ",
                 $"Insert Sequential ",
                 $"Journal",
                 $"Marker ",
                 $"Name Delimiters ",
                 $"Quit",
                 $"Reset",
                 $"Status",
                 $"Unlock File ",
                 $"Viewchange",
                0);
         = 'A:  %assimilate%
            BEGIN
            echo ($"Assimilate");
            ON SIGNAL
               =sighelp:
                  BEGIN
                  *lit* ← EOL,
                     " (structure) S,B,G, or P (to) ADDR CA", EOL,
                     " (from) ADDR CA LEVADJ CA VIEWSPEC CA";
                  typeas(slit);
                  RETURN;
                  END;
               ELSE;
            prompt ($" Structure ");
            CASE inpcuc() OF
               ='S:
                  BEGIN
```

```
            echo($"Statement");
            strtyp ← stmtv;
            END;
         ='B:
            BEGIN
            echo($"Branch");
            strtyp ← brnchv;
            END;
         ='G:
            BEGIN
            echo($"Group");
            strtyp ← groupv;
            END;
         ='P:
            BEGIN
            echo($"Plex");
            strtyp ← plexv;
            END;
         =CD:  GOTO STATE;
         ='?:
            BEGIN
            tyhelp(1); %type structural entities%
            REPEAT;
            END;
      ENDCASE
            BEGIN
            typech('?);
            REPEAT;
            END;
   crlf();
   prompt ($"to ");
   tbug ($stid);
   prompt ($" from ");
   CASE strtyp OF
      =stmtv, =brnchv:
            BEGIN
            tbug ($stid1);
            stid2 ← stid1;
            END;
      =groupv:
            BEGIN
            tbug ($stid1);
            prompt($" ");
            tbug ($stid2);
            stid1 ← <STRMNP, grptst> (stid1, stid2 : stid2);
            END;
      =plexv:
            BEGIN
            tbug ($stid1);
            stid1 ← <STRMNP, plxset> (stid1 : stid2);
            END;
      ENDCASE error();
   levadj (stid, $levj);
   savsp1 ← tda.davspec;  %save viewspecs to restore later%
   savsp2 ← tda.davspc2;
   getvsp();
```

```
      treslev (stid1);
      IF strtyp = stmtv THEN tda.davspec.vslev ← getlev (stid1);
      xea (stid, stid1, stid2, $levj, tda.davspec, tda.davspc2,
      tda.dausqcod, tda.dacacode);
      tda.davspec ← savsp1;  %restore viewspecs%
      tda.davspc2 ← savsp2;
      bl ← stid;
      bl[1] ← 1;
      mvcbl();
      freflnt();
      END;
   = 'B: %browse mode%
      BEGIN
      echo($"Browse ");
      CASE inpcuc() OF
         =CD: GOTO STATE;
         =CA, ='E:
            BEGIN
            echo($"Enter ");
            xebe(tda.dacsp.stfile);
            END;
         ='L :
            BEGIN
            echo($"Leave");
            echon();
            getctc();
            echo($" Keep Changes? ");
            echoff();
            xebl(answer(), tda.dacsp.stfile);
            END;
         = '?:
            BEGIN
            *lit* ← EOL,
               "Enter ", EOL,
               "Leave ";
            typeas($lit);
            RETURN;
            END;
      ENDCASE
            BEGIN
            typech('?);
            REPEAT;
            END;
      END;
   = 'C: %content analyser, Catalog Numbers%
      BEGIN
      echo($"C");
      CASE inpcuc() OF
         ='A: %Catalog Numbers%
            catnms();
         ='O: %Content Analyser%
            BEGIN
            echo($"ontent Analyzer ");
            ON SIGNAL
               =sighelp:
                  BEGIN
```

```
                typeas($" (at) ADDR CA");
                RETURN;
                END;
             ELSE;
          tbug($bl);
          IF (errs ← xgpc ($bl, &tda)) > 0 THEN
             BEGIN
             *str* ← STRING(errs), " error(s):  Type CA.";
             typeas ($str);
             clrbuf (0);  % clear input buffer %
             getctc();
             END;
          END;
       =sighelp:
          BEGIN
          *lit* ←EOL,
             "CAtalog Numbers", EOL,
             "COntent Analyzer ";
          typeas($lit);
          GOTO STATE;
          END;
       ENDCASE twohelp($"Catalog Numbers",  $"Content
       Analyser", 0);
    END;
= 'D: %Device Specification%
    BEGIN
    echo($"Device Specification");
    echo($" not implemented");
    END;
= 'E: %Execute Edit COMMAND%
    BEGIN
    echo($"Edit ");
    ON SIGNAL
       =sighelp:
          BEGIN
          *lit* ←  " (at) ADDR CA (CR) EDIT STUFF";
          typeas($lit);
          RETURN;
          END;
       ELSE;
    tbug($bl);
    editx($bl);
    tda.dacsp ← xrs(uselit, $bl, $bl, $lit);
    tda.daccnt ← 1;
    IF curchr = C. THEN trepstructure();
    RETURN;
    END;
= 'F:  %File Verify %
    BEGIN
    echo($"File ");
    todvfy();
       %goes to state%
    END;
= 'I: %insert sequential, identification system%
    BEGIN
    echo($"I");
```

```
CASE inpcuc() OF
   ='N: %insert sequential%
      BEGIN
      echo($"nsert Sequential from ");
      CASE inpcuc() OF
         ='A:
            BEGIN
            echo($"Assembler ");
            filtyp ← macfil;
            END;
         ='9:
            BEGIN
            echo($"940 ");
            filtyp ← n40fil;
            END;
         ='T:
            BEGIN
            echo($"Tenex ");
            filtyp ← tenfil;
            END;
         = '?:
            BEGIN
            *lit* ← EOL,
             "Assembler file " , EOL,
             "940 file " , EOL,
             "Tenex file";
            typeas($lit);
            END;
         =CD: GOTO STATE;
         ENDCASE
            BEGIN
            typech('?);
            REPEAT;
            END;
      prompt($" at ");
      ON SIGNAL
         =sighelp:
            BEGIN
            typeas($" (at) ADDR CA LEVADJ (from file)
            FILENAME");
            RETURN;
            END;
         ELSE;
      tbug($bl);
      levadj(bl, $levj);
      ON SIGNAL
         = ofilerr:
            BEGIN
            crlf();
            typeas (sysmsg);
            clrbuf (0);    % clear the input buffer %
            crlf();
            EXIT CASE;
            END;
         ELSE;
      prompt ($" from File ");
```

```
            echon();
            *flnam* ← NULL;
            rdlit ($flnam, 0);
            echoff();
            xei (bl, $levj, $flnam, filtyp);
            END;
         ='D: %identification system%
            BEGIN
            echo($"dentification submode");
            IF lookc() # CA THEN REPEAT CASE;
            input();
            <JIDENT, tjidcontrol>();
            END;
         =CD: GOTO STATE;
         ENDCASE
            BEGIN
            todco('?);
            REPEAT CASE;
            END;
      END;
   = 'J: %Execute Journal%
      BEGIN
      echo($"Journal ");
      crlf();
      tej();
      END;
   = 'M: %Marker%
      BEGIN
      echo($"Marker ");
      CASE inpcuc() OF
         ='F:
            BEGIN
            echo($"Fix");
            crlf();
            <TCMNDS, tf>();
            END;
         ='L, ='S:
            BEGIN
            echo($"List");
            crlf();
            <AUXCOD, seemkr>(tda.dacsp.stfile);
            END;
         ='R:
            LOOP BEGIN
            echo($"Release marker");
            prompt($" named ");
            *stn* ← NULL;
            echon();
            rdlit ($stn, $rnmdel);
            UNTIL stn.L MOD 5 = 0 DO *stn* ← *stn*, 0;
            <AUXCOD, delmkn>(lcfile(), stn[1]);
            IF curchr = CA THEN EXIT;
            crlf();
            END;
         ='?:
            BEGIN
```

```
            *lit* ← EOL,
               "Fix", EOL,
               "List", EOL,
               "Release";
            typeas($lit);
            RETURN;
            END;
         ENDCASE error();
      END;
   = 'N: %Name delimiter%
      BEGIN
      echo($"Name delimiter ");
      CASE inpcuc() OF
         = 'D: %Name delimiter display%
            BEGIN
            echo($"Display");
            ON SIGNAL
               =sighelp:
                  BEGIN
                  typeas($" (at) ADDR CA");
                  RETURN;
                  END;
               ELSE;
            LOOP
               BEGIN
               prompt($" at ");
               tbug($bl);
               mvcbl();
               <AUXCOD, getnmdls>(bl, $str);
               crlf();
               typeas($str);
               IF curchr # 0. THEN EXIT;
               crlf();
               END;
            END;
         = 'S: %Name delimiter statement%
            BEGIN
            echo ($"Statement");
            tens1 ($xens);
            END;
         = 'B: %Name delimiter branch%
            BEGIN
            echo ($"Branch");
            tens1 ($xenb);
            END;
         = 'P: %Name delimiter plex%
            BEGIN
            echo ($"Plex");
            tens1 ($xenp);
            END;
         = 'G: %Name delimiter group%
            BEGIN
            echo ($"Group");
            tens2 ($xeng);
            END;
         = '?:
```

```
            BEGIN
            *lit* ←
               EOL,
               "Display ";
            typp(1);
            typeas($lit);
            END;
         = SP: REPEAT CASE;
         ENDCASE error();
      END;
   = 'O: %ownership%
      BEGIN
      echo($"Ownership");
      getctc();
      crlf();
      xeo();
      END;
   = 'Q: %quit%
      BEGIN
      echo($"Quit");
      getctc();
      crlf();
      xeq();
      END;
   = 'R: %Reset%
      BEGIN
      echo($"Reset");
      getctc();
      prompt($" Really? ");
      IF answer() THEN
         BEGIN
         IF tda.daempty OR tda.dacsp = endfil THEN error();
         xer(&tda);
         tda.daccnt ← 1;
         END;
      END;
   %New Command = 'S: %% execute status %%
      tes();%
   ='S: %Old Command%
      BEGIN
      todco('S);
      CASE inpcuc() OF
         ='T: tes(); %Status%
         ='E: changcom($"Secondary Distribution", 0);
         ENDCASE twohelp($"Status",
            $"Secondary Distribution",
            0);
      END;
   = 'U:
      BEGIN
      echo($"Unlock File ");
      *lit* ← NULL;
      filnam(tda.dacsp.stfile, $lit);
      prompt($lit);
      getctc();
      prompt($" Really? ");
```

```
      IF answer() THEN <NLSEX, xeu>(tda.dacsp.stfile);
      END;
    = 'V: %viewchange%
      BEGIN
      echo($"Viewchange");
      viewch();
      END;
    ='W:
      BEGIN
      echo($"Write Access ");
      LOOP
         BEGIN
         CASE inpcuc() OF
            ='E:
               BEGIN
               echo($"Enable");
               IF lookc() = SP THEN
                  BEGIN
                  prompt($" For File: ");
                  *lit* ← NULL;
                  fileno ← open(0, $lit);
                  END
               ELSE fileno ← 0;
               enablaccess(fileno, gacctype());
               END;
            ='D:
               BEGIN
               echo($"Disable");
               IF lookc() =  SP THEN
                  BEGIN
                  prompt($" For File: ");
                  *lit* ← NULL;
                  fileno ← open(0, $lit);
                  END
               ELSE fileno ← 0;
               disabaccess(fileno, gacctype());
               END;
            ='S:
               BEGIN
               echo($"Set");
               IF lookc() = SP THEN
                  BEGIN
                  prompt($" For File: ");
                  *lit* ← NULL;
                  fileno ← open(0, $lit);
                  END
               ELSE fileno ← tda.dacsp.stfile;
               setaccess(fileno, gacctype());
               IF fileno # tda.dacsp.stfile THEN close(fileno);
               END;
            =CA, =CD: EXIT;
            ENDCASE error();
         crlf();
         END;
      END;
    ='X: %colsort%
```

```
            BEGIN
            echo($"Xcolsort ");
            getctc();
            colsrt();
            END;
         ENDCASE error();
      GOTO STATE;
      RETURN;
      END.

(catnums)PROC;
      %Catalog Numbers%
      LOCAL litadr, numcnt;
      echo($"atalog Number System");
      STATE ← cnmstate, spec;
      thearald('#);
      CASE inpcuc() OF
         ='A: BEGIN
            echo($"Assign number(s) for collection ");
            CASE inpcuc() OF
               ='N:
                  BEGIN
                  echo($"NIC ");
                  litadr ← $" NIC ";
                  END;
               ='R:
                  BEGIN
                  echo($"RINS ");
                  litadr ← $" RINS ";
                  END;
               ='X:
                  BEGIN
                  echo($"Xdoc ");
                  litadr ← $" XDOC ";
                  END;
               ='S:
                  BEGIN
                  echo($"Special ");
                  litadr ← $" SPECIAL ";
                  END;
            ENDCASE error();
            IF (numcnt ←  gttnum(1)) > 20 THEN
               err($"Maximum numbers = 20");
            IF curchr # CA THEN getctc();
            numstid ← openlock(0, jflname($"tcnumbers"));
            ON SIGNAL ELSE closeu(numstid.stfile := 0);
            WHILE (numcnt ← numcnt-1) >= 0 DO BEGIN
               crlf();
               %now assign a number, and mark it used%
                  getcnum($initsr, $stno, litadr, 2, numstid);
                   typeas($stno);
               END;
            closeu(numstid.stfile := 0);
            END;
         ='P:
            BEGIN
```

```
        echo($"Pre-assign Journal Number(s) ");
        IF (numcnt ← gttnum(1)) > 20 THEN
           err($"Maximum numbers = 20");
        IF curchr # CA THEN getctc();
        numstid ← openlock(0, jflname($"tcnumbers"));
        ON SIGNAL ELSE closeu(numstid.stfile := 0);
        WHILE (numcnt ← numcnt-1) >= 0 DO
           BEGIN
           crlf();
           getcnum($initsr, $stno, $" JOURNAL ", 1, numstid);
           typeas($stno);
           END;
        closeu(numstid.stfile := 0);
        END;
     ='R:
        pasrfc(); %Pre-assigned RFC Number%
     ='Q:
        BEGIN
        echo($"Quit");
        getctc();
        reset();
        END;
  ENDCASE error();
  GOTO STATE;
   END.
(tej) PROCEDURE; %Execute Journal%
  LOCAL STRING passwd[5];
  LOCAL char;
  STATE ← tejstate, spec;
  thearald('&);
  CASE inpcuc() OF
     = 'S: %Submit%
        BEGIN
        echo($"Submit ");
        <NLS, JOCTL, jcontrol>();
        END;
     = 'C: %Catalog Cleanup%
        BEGIN
        echo($"Catalog update");
        crlf();
        typeas($" Password--");
        *passwd* ← NULL;
        UNTIL passwd.L = 3 DO
           IF (char ← inpcuc()) = CD THEN
              GOTO STATE
           ELSE *passwd* ← *passwd*, char;
        IF *passwd* # "JPD" THEN err($"Nope");
        IF jdebug THEN
           BEGIN
           crlf();
           typeas($"Experimental Journal System");
           END;
        getctc();
        crlf();
        updtjo();
        END;
```

```
      = 'D: %Distribute Document%
         BEGIN
         echo($"Distribute Document #");
         LOOP
            BEGIN
            IF jdebug THEN
               BEGIN
               crlf();
               typeas($"(Experimental Journal System) #");
               END;
            CASE lookc() OF
               =CA, =CD: GOTO STATE;
               IN ['0, '9]: jdistd();
               ENDCASE
                  BEGIN
                  input();
                  typeas($"  ?");
                  END;
            END;
         END;
      = 'H: %Hard copy distribution%
         BEGIN
         echo($"Hard Copy Journal Distribution");
         hcdex();
         END;
      = 'Q: %Quit%
         BEGIN
         echo ($"Quit");
         getctc();
         nlsrst();
         END;
      = '?:
         help(
              $"Submit ",
              $"Distribute Document",
              $"Quit",
             0);
      ENDCASE error();
      GOTO STATE;
   RETURN; END.

(tes) PROCEDURE; %Execute Status command%
   echo($"tatus ");
   CASE inpcuc() OF
      ='V:  %viewspecs%
         BEGIN
         echo($"Viewspecs ");
         curvsp(&tda, $lit);
         END;
      ='L:  %Link stack%
         BEGIN
         echo($"Link stack ");
         <NLS, NLSEX, xesl>($lit);
         END;
      ='F:  %File%
         BEGIN
```

```
            echo($"File ");
            <NLS, NLSEX, xesf> (tda.dacsp.stfile, $lit);
            END;
         ='?:
            BEGIN
            *lit* ←
               "Viewspecs ", EOL,
               "File ", EOL,
               "Link stack ";
            END;
         ENDCASE error();
      crlf();
      typeas($lit);
      RETURN;
      END.

(curvsp) PROCEDURE(da, astrng);
      LOCAL vspc;
      REF da, astrng;
      vspc ← da.davspec;
      %level%
         *astrng* ← "L=";
         IF vspc.vslev = 63 THEN *astrng* ← *astrng*, "ALL"
         ELSE *astrng* ← *astrng*, STRING(vspc.vslev);
      %truncation%
         *astrng* ← *astrng*, ", T=";
         IF vspc.vstrnc = 63 THEN *astrng* ← *astrng*, "ALL"
         ELSE *astrng* ← *astrng*, STRING(vspc.vstrnc);
         *astrng* ← *astrng*, ", ";
      %branch only stuff%
         IF vspc.vsbrof THEN *astrng* ← *astrng*, 'g
         ELSE IF vspc.vsplxf THEN *astrng* ← *astrng*, 'l
         ELSE *astrng* ← *astrng*, 'h;
      %content analysis%
         IF vspc.vscapf THEN *astrng* ← *astrng*, 'i
         ELSE IF vspc.vscakf THEN *astrng* ← *astrng*, 'k
         ELSE *astrng* ← *astrng*, 'j;
      %frozen stats%
         IF vspc.vsfrzf THEN *astrng* ← *astrng*, 'o
         ELSE *astrng* ← *astrng*, 'p;
      %formatter on/off%
         IF vspc.vsdaft THEN *astrng* ← *astrng*, 'u
         ELSE *astrng* ← *astrng*, 'v;
      %blank lines%
         IF vspc.vsblkf THEN *astrng* ← *astrng*, 'y
         ELSE *astrng* ← *astrng*, 'z;
      %indenting%
         IF vspc.vsindf THEN *astrng* ← *astrng*, 'A
         ELSE *astrng* ← *astrng*, 'B;
      %names%
         IF vspc.vsnamf THEN *astrng* ← *astrng*, 'C
         ELSE *astrng* ← *astrng*, 'D;
      %stat nums left/right%
         IF vspc.vsstnr THEN *astrng* ← *astrng*, 'G
         ELSE *astrng* ← *astrng*, 'H;
      %signature%
```

```
        IF vspc.vsidtf THEN *astrng* ← *astrng*, 'K
        ELSE *astrng* ← *astrng*, 'L;
      %markers%
        IF vspc.vsmkrf THEN *astrng* ← *astrng*, 'M
        ELSE *astrng* ← *astrng*, 'N;
      %user sequence generator%
        IF vspc.vsusqf THEN *astrng* ← *astrng*, 'O
        ELSE *astrng* ← *astrng*, 'P;
      RETURN END.

    (gacctype)PROC;
      CASE inpcuc() OF
        ='N:
          BEGIN
          echo($" Normal files");
          RETURN(normlaccess);
          END;
        ='J:
          BEGIN
          echo($" Journal files");
          IF inpcuc() # '' THEN error();
          RETURN(jrnlaccess)
          END;
        ENDCASE;
        error();
      END.

%.....Execute Edit Routines.......%

    (editx) PROCEDURE (ptr);
      %-----------------%
      LOCAL
        char, %last character read%
        number,
        insrtf;
      REF ptr;
      CCPOS SF(ptr);
      *lit* ← NULL; %INITIALISE STRING%
      crlf();
      echoff();
      insrtf ← FALSE;
      %deactivate ↑O and ↑S interrupts%
        r1 ← 4B5; r2 ← 24B10; !JSYS dic;
        r1 ← 15;  !JSYS 140B;  %deassign terminal code%
        r1 ← 19;  !JSYS 140B;  %deassign terminal code%
      LOOP
        CASE char ← input() OF
          =BC %↑H%, =$ctla: %backspace character%
            BEGIN
            bkc($lit);
            todco(BC);
            END;
          =CA %↑D%, =C. %↑B%: %copy remainder of old to new and
          terminate alter%
            BEGIN
            todco(char);
```

```
        copych(2000, typefg);
        EXIT;
        END;
    =$ctle: %change source of text to or from keyboard and type
    < or >%
        IF insrtf THEN
            BEGIN
            insrtf ← FALSE;
            typech('>);
            END
        ELSE
            BEGIN
            insrtf ← TRUE;
            typech('<);
            END;
    =$ctlf: %copy 1 chr and type%
         IF copych(1) THEN EXIT;
    =$ctlg: %skip up thru c and type percent ptr%
        IF (number ← search(input())) THEN skip(number)
        ELSE typeas($" ?");
    =$ctln: %backspace character in old and new%
        BEGIN
        bkc($lit);
        typech('↑);
        swork1 ← MAX(1, swork1-1);
        fechc1(forward, $swork);
        END;
    =$ctlo: %copy to c%
        IF (number ← search(input())) THEN copych(number-1)
        ELSE typeas($" ?");
    =$ctlp: %skip up to c and type percent sign %
        IF (number ← search(input())) THEN skip(number-1)
        ELSE typeas($" ?");
    =$ascbst %↑Q%: %backspace statement in old and new%
        BEGIN
        todco($ascbst);
        crlf();
        *lit* ← NULL;
        CCPOS SF(ptr);
        END;
    =$ctlr: %retype line up to this point%
        BEGIN
        crlf();
        typeas($lit);
        END;
    =$ctls: %skip one character on old statement--type percent%
         IF skip(1) THEN EXIT; %end of statement%
    =$ctlu: %copy thru end of old statement%
        copych(2000);
    =BW, =$ctlw: %backspace word in new%
        BEGIN
        <NLS, UTILTY, bkw>($lit);
        todco(BW);
        END;
    =CD %↑X%:
        BEGIN
```

```
            edixqt();
            GOTO STATE;
            END;
         =$ctly: %repeat alter%
            BEGIN
            copych(2000, typefg);
            xrs(uselit, &ptr, &ptr, $lit);
            tda.dacsp ← ptr;
            tda.daccnt ← 1;
            REPEAT($ascbst);
            END;
         =$ctlz: %copy thru c%
            IF (number ← search(input())) THEN copych(number)
            ELSE typeas($" ?");
         =$ascalt: % terminate alter%
            EXIT;
         ENDCASE %other character..insrtf into new%
            BEGIN
            todco(char);
            *lit* ← *lit*, char;
            IF NOT insrtf THEN char ← READC;
            END;
   edixqt();
   RETURN;
   END.

(edixqt) PROCEDURE;
   %reassign the terminal codes%
      r1 ← 230000001B; %↑S%
      !JSYS ati;
      r1 ← 170000003B; %↑O%
      !JSYS ati;
   r1 ← 4B5;
   r2 ← 24B10; %activate ↑S, ↑O%
   !JSYS aic;
   RETURN END.

(copych) PROCEDURE (number);
   %copy 'number' characters from old to new%
   %------------------%
   LOCAL char;
   UNTIL (number := number - 1) <= 0 DO
      CASE char ← READC OF
         =ENDCHR: RETURN(TRUE);
         ENDCASE
            BEGIN
            *lit* ← *lit*, char;
            typech(char);
            END;
   RETURN(FALSE);
   END.

(search) PROCEDURE (target);
   %search old statement for character in target.  return number of
   characters away from current swork setting if found, 0 if not
   found%
```

```
    %-----------------%
    LOCAL count, save, last, char;
    count ← 1;
    save ← swork[1];
    LOOP
       CASE READC OF
          = ENDCHR: RETURN(FALSE);
          = target:
             BEGIN
             swork[1] ← save;
             fechcl(forward, $swork);
             RETURN(count);
             END;
          ENDCASE BUMP count;
    END.

(skip) PROCEDURE (number);
    %skip a number of characters in old and type percent signs%
    %-----------------%
    LOCAL last, char;
    UNTIL (number ← number - 1) < 0 DO
       CASE READC OF
          =ENDCHR: RETURN(TRUE);
          ENDCASE typech('%');
    RETURN(FALSE);
    END.

%.....viewchange command code.....%

(viewch) PROCEDURE;  %viewchange command language code%
    %-----------------%
    LOOP
       BEGIN
       echoff();
       crlf();
       prompt($"  ");
       CASE inpcuc() OF
          ='C:  %character define%
             chrdfn();
          ='F:  %feedback%
             fedbck();
          ='S:  %shift case%
             scase();
          ='T:  %text area%
             tarea();
          ='/:  %/ command characters%
             BEGIN
             todco('/);
             prompt($" characters ");
             tslshchars ← sttnum(tslshchars);
             END;
          = CA: EXIT;
          = CD: GOTO STATE;
          = '?:
             BEGIN
             *lit* ← EOL,
```

```
                "/ characters", EOL,
                "Character define", EOL,
                "Feedback", EOL,
                "Shift case", EOL,
                "Text area";
             typeas($lit);
             RETURN;
             END;
          = SP : REPEAT;
          ENDCASE typech('?);
       END;
    RETURN
    END.

(chrdfn) PROCEDURE; %char define command code--viewch support%
    %------------------%
    LOCAL newchr, oldchr, type;
    echo($"Character set");
    LOOP
       BEGIN
       echoff();
       crlf();
       prompt($"  ");
       CASE inpcuc() OF
          ='D: BEGIN
             echo($"Define ");
             echon();
             oldchr ← rawchr();
                %read a raw character%
             prompt($" as ");
             CASE newchr ← input() OF
                = SP:
                   CASE inpcuc() OF
                      ='C: CASE inpcuc() OF
                         ='A: newchr ← CA;
                         ='D: newchr ← CD;
                         ='.: newchr ← C.;
                         =CD: GOTO STATE;
                         ENDCASE
                            BEGIN
                            typech('?);
                            REPEAT CASE 3;
                            END;
                      ='B: CASE inpcuc() OF
                         ='C: newchr ← BC;
                         ='S: newchr ← $ascbst;
                         ='W: newchr ← BW;
                         =CD: GOTO STATE;
                         ENDCASE
                            BEGIN
                            typech('?);
                            REPEAT CASE 3;
                            END;
                      ='L: BEGIN
                         echo($"iteral Escape");
                         todlit ← oldchr;
```

```
                    EXIT;
                    END;
                =='N: BEGIN
                    echo($"ull");
                    newchr ← nullch;
                    END;
                =='S: BEGIN
                    echo($"witch");
                    newchr ← $asctsw;
                    END;
                =='T: BEGIN
                    echo($"ab");
                    newchr ← TAB;
                    END;
                =SP: BEGIN %read untranslated character%
                    newchr ← rawchr();
                    END;
                =CD: GOTO STATE;
                ENDCASE BEGIN
                    typech('?);
                    EXIT;
                    END;
            =CD: GOTO STATE;
            IN [0, 37B], =BC, =BW, =CA, =C., =$ascbst, =177B:
                NULL;
            ENDCASE
                BEGIN
                typech('?);
                REPEAT CASE;
                END;
        %by here, newchr set up%
        %update input translate table%
            *trnsli*[oldchr] ← newchr;
        %update output translate table%
            oldchr ← newchr;
            LOOP
                BEGIN
                prompt($" Echo as ");
                IF (newchr ← rawchr()) = SP THEN
                    CASE inpcuc() OF
                        = 'N:
                            BEGIN
                            echo($"ull");
                            newchr ← nullch;
                            END;
                        =CA: NULL;
                        =CD: GOTO STATE;
                        ENDCASE
                            BEGIN
                            typech('?);
                            REPEAT LOOP;
                            END;
                EXIT LOOP;
                END;
            *trnslo*[oldchr] ← newchr;
        crlf();
```

```
            prompt($"  ");
            END;
         = CA: EXIT;
         = CD: GOTO STATE
         = '?:
            BEGIN
            *lit* ← EOL, "Define";
            typeas($lit);
            RETURN;
            END;
         ENDCASE typech('?);
      END;
   RETURN;
   END.

(fedbck) PROCEDURE;  %feedback--viewch support%
   %-----------------%
   echo($"Feedback");
   LOOP
      BEGIN
      echoff();
      crlf();
      prompt($"  ");
      CASE inpcuc() OF
         ='L: BEGIN
            echo($"Levadj numbers");
            lvdjnm ← NOT lvdjnm;
            END;
         ='C: BEGIN
            echo($"Characters ");
            feedbk ← sttnum(feedbk);
            END;
         ='I: BEGIN
            echo($"Indenting ");
            fedind ← sttnum(fedind);
            END;
         = '?:
            BEGIN
            *lit* ← EOL,
               "Levadj numbers ", EOL,
               "Characters ", EOL,
               "Indenting ";
            typeas($lit);
            END;
         = CA: EXIT;
         = CD: GOTO STATE;
      ENDCASE typech('?);
      END;
   RETURN;
   END.

(scase) PROCEDURE;  %shift case--viewch support%
   %-----------------%
   LOCAL shiftc, type, newchr, oldchr;
   echo($"Shift Case");
   LOOP
```

```
BEGIN
echoff();
crlf();
prompt($"  ");
CASE inpcuc() OF
   = 'L: BEGIN
      echo($"Lower Case ");
      shiftc ← 0;
      END;
   = 'U: BEGIN
      echo($"Upper Case ");
      shiftc ← 1;
      END;
   = 'C: BEGIN
      echo($"Control Case ");
      shiftc ← 2;
      END;
   ='O: BEGIN
      echo($"Off");
      pshift ← tshift ← 3;
      REPEAT LOOP;
      END;
   = '?:
      BEGIN
      *lit* ← EOL,
         "Control Case ", EOL,
         "Lower Case ", EOL,
         "Upper Case ", EOL,
         "Off ";
      typeas($lit);
      END;
   = CA: EXIT;
   = CD: GOTO STATE;
   ENDCASE
      BEGIN
      typech('?);
      EXIT LOOP;
      END;
 CASE inpcuc() OF
   = 'C: BEGIN
      echo($"Character Shift ");
      type ← $cshft0;
      END;
   = 'W: BEGIN
      echo($"Word Shift ");
      type ← $wshft0
      END;
   = 'P: BEGIN
      echo($"Permanent Shift ");
      type ← $pshft0;
      END;
   = '?:
      BEGIN
      *lit* ← EOL,
         "Character Shift ", EOL,
         "Word Shift ", EOL,
```

```
                "Permanent Shift ";
            typeas($lit);
            END;
         ENDCASE
            BEGIN
            typech('?);
            EXIT LOOP;
            END;
      shiftc ← type + shiftc;
      echon();
      prompt($" as ");
      CASE newchr ← input() OF %check for NULL%
         = SP: CASE inpcuc() OF
            = 'N: BEGIN
               echo($"ull");
               newchr ← nullch;
               END;
            = '?:
               BEGIN
               *lit* ← EOL,
                  "Null ";
               typeas($lit);
               END;
            ENDCASE
               BEGIN
               typech('?);
               EXIT LOOP;
               END;
         =CA, =CD:
            BEGIN %not allowed%
            typech('?);
            EXIT;
            END;
         ENDCASE newchr ← newchr; %for L10 bug%
      [shiftc] ← newchr;
      END;
   RETURN;
   END.

(tarea) PROCEDURE;  %Text Area command code--viewch support%
   %-----------------%
   LOCAL fv, char, ntab, otab, tabr2, tabr3, tabr4, count;
   echo($"Text area");
   LOOP
      BEGIN
      echoff();
      crlf();
      prompt($"   ");
      CASE inpcuc() OF
         ='C:  %columns%
            BEGIN
            echo($"Columns ");
            tda.damcol ←
               sttnum(tda.damcol/tda.dahinc)*tda.dahinc;
            END;
         ='T:  %tabs%
```

```
BEGIN
echo($"Tabs: ");
%read current tab setting in monitor%
   r1 ← 18M;
   !JSYS gtabs;
   tabr2 ← r2;
   tabr3 ← r3;
   tabr4 ← r4;
CASE (char ← input()) OF
   IN ['0, '9]:
      fv ← CASE char OF
         = '0: datab0;
         = '1: datab1;
         = '2: datab2;
         = '3: datab3;
         = '4: datab4;
         = '5: datab5;
         = '6: datab6;
         = '7: datab7;
         = '8: datab8;
         ENDCASE datab9;
   = CA: EXIT CASE 2;
   = CD: GOTO STATE;
   ENDCASE
      BEGIN
      typech('?);
      REPEAT CASE;
      END;
tda.fv ← ntab ←
   sttnum(otab ← (tda.fv/tda.dahinc))*tda.dahinc;
ntab ← ntab/tda.dahinc;
%give new tab settings to monitor%
   CASE otab OF
      IN [0, 35]:
         BEGIN
         otab ← otab .X -1;
         r1 ← 1B35;
         !LSH r1,otab;
         !SETCM r1,r1;
         tabr2 ← tabr2 .A r1;
         END;
      IN [36, 71]:
         BEGIN
         otab ← (otab - 36) .X -1;
         r1 ← 1B35;
         !LSH r1,otab;
         !SETCM r1,r1;
         tabr3 ← tabr3 .A r1;
         END;
      ENDCASE
         BEGIN
         otab ← (otab - 72) .X -1;
         r1 ← 1B35;
         !LSH r1,otab;
         !SETCM r1,r1;
         tabr4 ← tabr4 .A r1;
```

```
            END;
         CASE ntab OF
            IN [0, 35]:
               BEGIN
               ntab ← ntab .X -1;
               r1 ← 1B35;
               !LSH r1,ntab;
               tabr2 ← tabr2 .V r1;
               END;
            IN [36, 71]:
               BEGIN
               ntab ← (ntab - 36) .X -1;
               r1 ← 1B35;
               !LSH r1,ntab;
               tabr3 ← tabr3 .V r1;
               END;
            ENDCASE
               BEGIN
               ntab ← (ntab - 72) .X -1;
               r1 ← 1B35;
               !LSH r1,ntab;
               tabr4 ← tabr4 .V r1;
               END;
         r1 ← 18M;
         r2 ← tabr2;
         r3 ← tabr3;
         r4 ← tabr4;
         !JSYS stabs;
      END;
   ='I:  %indenting%
      BEGIN
      echo($"Indenting ");
      tda.daind ←
         sttnum(tda.daind/tda.dahinc)*tda.dahinc;
      END;
   ='L:  %lines printed per page%
      BEGIN
      echo($"Lines/Page ");
      tda.damrow ←
         sttnum(tda.damrow/tda.davinc)*tda.davinc;
      END;
   ='R:  %rows per total page%
      BEGIN
      echo($"Rows/Page ");
      tda.davinc ← (tda.dabottom-tda.datop)/
         sttnum(tda.dabottom-tda.datop/tda.davinc);
      END;
   = '?:
      BEGIN
      *lit* ← EOL,
         "Levadj numbers ", EOL,
         "Characters ", EOL,
         "Indenting ";
      typeas($lit);
      END;
   = CA: EXIT;
```

```
            = CD: GOTO STATE;
              ENDCASE typech('?);
          END;
       RETURN;
       END.

    (sttnum) PROCEDURE (value);  %get number from user--uses gttnum%
       %------------------%
       LOCAL nwvalu;
       prompt($"= ");
       typeno(value, 0); %type the old value%
       prompt($"  ");
       IF (nwvalu ← gttnum(value)) <= -1 THEN nwvalu ← 0;
       CASE inpcuc() OF
          = CA:  RETURN(nwvalu);
          = CD:  GOTO STATE;
          ENDCASE
             BEGIN
             typech('?);
             GOTO STATE;
             END;
       END.

%.....TNLS Goto commands parser........%

    (tgoto) PROCEDURE;
       LOCAL jfn, guflag, num, write, bintyp;
       LOCAL STRING flnam[50];
       write ← 0; bintyp ← 2;
       echo($"Goto ");
       CASE inpcuc() OF
          = 'B:   %Baseline%
             <BASELN, tgb> ();
          = 'M:  %Merge%
             BEGIN
             echo($"Merge ");
             CASE inpcuc() OF
                = 'B:  %Branch%
                   BEGIN
                   echo ($"Branch");
                   tgms1($xgmeb);
                   END;
                = 'G:  %Group%
                   BEGIN
                   echo ($"Group");
                   tgms2($xgmeg);
                   END;
                = 'P:  %Plex%
                   BEGIN
                   echo ($"Plex");
                   tgms1($xgmep);
                   END;
                = CA:  REPEAT ('P);
                ENDCASE error();
             END;
          = 'P:   %user Programs%
```

```
    <USRPGM, tgp> ();
= 'S:  %Sort%
    BEGIN
    echo($"Sort ");
    CASE inpcuc() OF
       = 'B:  %Branch%
          BEGIN
          echo ($"Branch");
          tgss1($xgsob);
          END;
       = 'G:  %Group%
          BEGIN
          echo ($"Group");
          tgss2($xgsog);
          END;
       = 'P:  %Plex%
          BEGIN
          echo ($"Plex");
          tgss1($xgsop);
          END;
       = CA:  REPEAT ('P);
       ENDCASE error();
    END;
= 'U:  %Use measurement%
    BEGIN
    echo($"Use Measurement Begin ");
    guflag ← TRUE;
    LOOP
       CASE input() OF
          ='B:
             BEGIN
             echo($"Begin ");
             guflag ← TRUE;
             END;
          ='E:
             BEGIN
             echo($"End ");
             guflag ← FALSE;
             END;
          =CA: EXIT LOOP;
          =CD: GOTO STATE;
          ENDCASE typech('?);
    IF guflag THEN
       IF NOT msflag THEN
          BEGIN
          LOOP
             BEGIN
             prompt($" File ");
             echon();
             *flnam* ← NULL;
             rdlit ($flnam, 0);
             IF NOT (jfn ← lgetjfn (0, $flnam, $nlsext, gtjoif,
             $lit)) THEN
                typeas ($lit)
             ELSE EXIT LOOP 1;
             crlf();
```

```
                    echoff();
                    END;
                 crlf();
                 prompt($" Update Interval (minutes) ");
                 num ← 0;
                 WHILE (num ← gttnum(num)) = 0 DO todco('?);
                 <AUXCOD, initms>(jfn, num);
                 END
              ELSE dismes(2, $"Measurements Being Recorded ")
           ELSE <AUXCOD, closms>();
           GOTO STATE;
           END;
        ENDCASE error();
    RETURN; END.

(tgss1) PROCEDURE(proc);
    REF proc;
    ON SIGNAL
       = sighelp :
          BEGIN
          typeas($" (at) ADDR CA CA");
          RETURN;
          END;
       ELSE;
    LOOP
       BEGIN
       prompt($" at ");
       tbug($bl);
       prompt($" ok?");
       getctc();
       mvcbl();
       proc($bl);
       IF curchr # 0. THEN EXIT;
       crlf();
       END;
    RETURN END.

(tgss2) PROCEDURE(proc);
    REF proc;
    ON SIGNAL
       = sighelp :
          BEGIN
          typeas($" (at) ADDR CA ADDR CA CA");
          RETURN;
          END;
       ELSE;
    LOOP
       BEGIN
       prompt($" at ");
       tbug($bl);
       prompt($" ");
       tbug($b2);
       prompt($" ok?");
       getctc();
       mvcbl();
       proc($bl, $b2);
```

```
      IF curchr # 0. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tgms1) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      =sighelp:
         BEGIN
         typeas($" (into) ADDR CA (from) ADDR CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      prompt($" into ");
      tbug($b1);
      prompt($" from ");
      tbug($b2);
      prompt($" ok?");
      getctc();
      mvcb1();
      proc($b1, $b2);
      IF curchr # 0. THEN EXIT;
      crlf();
      END;
   RETURN END.

(tgms2) PROCEDURE(proc);
   REF proc;
   ON SIGNAL
      =sighelp:
         BEGIN
         typeas($" (into) ADDR CA ADDR CA (from) ADDR CA ADDR CA");
         RETURN;
         END;
      ELSE;
   LOOP
      BEGIN
      prompt($" into ");
      tbug($b1);
      prompt($" ");
      tbug($b2);
      prompt($" from ");
      tbug($b3);
      prompt($" ");
      tbug($b4);
      prompt($" ok?");
      getctc();
      mvcb1();
      proc($b1, $b2, $b3, $b4);
      IF curchr # 0. THEN EXIT;
      crlf();
      END;
   RETURN END.
```

```
%.....Name delimiter support routines........%

   (tens1) PROCEDURE (proc);
      LOCAL STRING str1[4], strr[4];
      LOCAL dlleft, dlrght;
      REF proc;
      ON SIGNAL
         = sighelp :
            BEGIN
            typeas($" (at) ADDR CA");
            RETURN;
            END;
         ELSE;
      LOOP
         BEGIN
         prompt($" at ");
         tbug($bl);
         crlf();
         prompt($"Left delimiter ");
         *str1* ← NULL;
         txtlit($str1);
         crlf();
         prompt($"Right delimiter ");
         *strr* ← NULL;
         txtlit($strr);
         prompt($" ok? ");
         getctc();
         todco(CA);
         dlleft ←
            IF str1.L =empty OR (FIND SF(*str1*) ("null"/"NULL")) THEN
            0
            ELSE *str1*[1];
         dlrght ←
            IF strr.L =empty OR (FIND SF(*strr*) ("NULL"/"null")) THEN
            0
            ELSE *strr*[1];
         proc($bl, dlleft, dlrght, tda);
         IF curchr # C. THEN EXIT;
         crlf();
         END;
      RETURN
      END.

   (tens2) PROCEDURE (proc);
      LOCAL STRING str1[4], strr[4];
      LOCAL dlleft, dlrght;
      REF proc;
      ON SIGNAL
         = sighelp :
            BEGIN
            typeas($" (at) ADDR CA");
            RETURN;
            END;
         ELSE;
      LOOP
```

TXTEDT

```
<NLS>TXTEDT.NLS;24, 2-JAN-72 18:09 BLP ;1, 16-NOV-1858 19:00 MSC ;
FILE txtedt   %  L1O  <REL-NLS>TXTEDT   %

% entity finding routines - known as delimiters %

    (cdr) PROCEDURE (bug,ptr1,ptr2);
       REF bug, ptr1, ptr2;
       FIND bug > ↑ptr1 CH ↑ptr2;
       RETURN;
       END.

    (idr) PROCEDURE (bug,ptr1,ptr2);
       REF bug, ptr1, ptr2;
       FIND bug < $NP ↑ptr1 bug > CH $NP ↑ptr2;
       RETURN;
       END.

    (tdr) PROCEDURE (bug1,bug2,ptr1,ptr2);
       REF bug1, bug2, ptr1, ptr2;
       IF POS SF(bug1) # SF(bug2) THEN err($"illegal entity")
       ELSE
          IF POS bug1 < bug2 THEN
             FIND bug1 ↑ptr1 bug2 > CH ↑ptr2
          ELSE
             FIND bug2 ↑ptr1 bug1 > CH ↑ptr2;
       RETURN;
       END.

    (ndr) PROCEDURE (bug,ptr1,ptr2);
       LOCAL dpoint;
       REF bug, ptr1, ptr2;
       dpoint ← FALSE;
       IF NOT FIND
          bug > (
          '. 1$D ↑ptr2 bug < $D ↑ptr1 /
          1$D ('. 1$D FS dpoint / FR dpoint) ↑ptr2
          1$D ('. 1$D FS dpoint / FR dpoint)
             bug < $D (FT dpoint / '. $D / TRUE) ↑ptr1)
       THEN err($"illegal number");
       RETURN;
       END.

    (wdr) PROCEDURE (bug,ptr1,ptr2);
       REF bug, ptr1, ptr2;
       FIND bug > CH $LD ↑ptr2 bug < $LD ↑ptr1;
       RETURN;
       END.

    (nmdr) PROCEDURE (bug,ptr1,ptr2);
       REF bug, ptr1, ptr2;
       FIND bug >
          $(LD / '- /'' ) ↑ptr2
          bug < $(LD / '- /'' ) ↑ptr1;
       RETURN;
       END.

    (vdr) PROCEDURE (bug,ptr1,ptr2);
```

```
    REF bug, ptr1, ptr2;
    FIND bug > CH $PT ↑ptr2 bug < $PT ↑ptr1;
    RETURN;
    END.

(ldr) PROCEDURE (bug,ptr1,ptr2);
    REF bug, ptr1, ptr2;
    IF NOT FIND
       bug > ['] / '>] ↑ptr2 < ['( / '< / '- '- > CH] ↑ptr1
    THEN err($"illegal link");
    RETURN;
    END.

% link parsing %  %in L10%

(lparse) PROCEDURE (bug,u1,u2,f1,f2,n1,n2,v1,v2,number);
    REF bug;
    FIND bug;
    fndend(number);      %find right end of proper link%
    vparse(v1,v2);           %parse view spec field%
    nfparse(n1,n2);          %parse statement name field%
    nfparse(f1,f2);          %parse file name field%
    uparse(u1,u2);           %parse user name field%
    RETURN;
    END.

(fndend) PROCEDURE (number); %find end of 'number'th link%
    UNTIL (number := number-1) = 0 DO
       IF NOT FIND > ['] / '>] THEN err($"illegal link");
    FIND < CH;
    RETURN;
    END.

(vparse) PROCEDURE (v1,v2);  %view-spec field parse%
    REF v1,v2;
    IF NOT FIND
       < $NP ↑v2 ↑v1
       [ ': > CH $NP ↑v1 < $NP CH /
       ( '( / '< / "--" / ', ) v2 ]
    THEN err($"illegal link");
    RETURN;
    END.

(nfparse) PROCEDURE (a1,a2); %statement name, file name parse%
    REF a1,a2;
    IF NOT FIND
       < $NP ↑a2
       [', > CH $NP ↑a1 < $NP CH /
       ('( / '< / '- '- > CH) > CH $NP ↑a1]
    THEN err($"illegal link");
    RETURN;
    END.

(uparse) PROCEDURE (u1,u2);  %user name parse%
    REF u1,u2;
    IF NOT FIND
```

```
      < $NP ↑u2 [('( / '< / '- '- > CH) > CH $NP ↑u1]
    THEN err($"illegal link");
    RETURN;
    END.

%routines for replace and transpose number%

  (trnchk) PROCEDURE (ptr1,ptr2,ptr3,ptr4);
    %moves pointers around for transpose number%
    LOCAL delt;
    delt ← [ptr2+1] - [ptr1+1] - [ptr4+1] + [ptr3+1];
    CASE delt OF
      < 0: rjshift(ptr1,delt);
      > 0: rjshift(ptr3,-delt);
      ENDCASE;
    RETURN;
    END.

  (rjshift) PROCEDURE (ptr,delt);
    REF ptr;
    FIND ptr <;
    LOOP
      IF (delt := delt+1) >= 0 OR
      (NOT FIND SP ↑ptr) THEN RETURN;
    END.

  (rjrepn) PROCEDURE (ptr1, ptr2, astring);
    LOCAL delt;
    REF astring;
    delt ← [ptr2+1] - [ptr1+1] - astring.L;
    CASE delt OF
      < 0: rjshift(ptr1,delt);
      > 0: BEGIN
        *num* ← *astring*;
        *astring* ← NULL;
        UNTIL (delt ← delt-1) < 0 DO *astring* ← *astring*,
        SP;
        *astring* ← *astring*, *num*;
        END;
      ENDCASE;
    RETURN;
    END.

% append %

  (staptx) PROCEDURE (a1, a2, astring);
    REF astring;
    ST a1 ← SF(a1) SE(a1), *astring*, SF(a2) SE(a2);
    RETURN;
    END.

% break %

  (brkst) PROCEDURE (a1, a2, astring);
    REF a1, astring;
    FIND a1 > CH $PT ↑p1 $NP ↑p2;
```

```
      cldtxt($p1, $p2);
      ST a2 ← *astring*, p2 SE(a1);
      ST a1 ← SF(a1) p1;
      RETURN;
      END.

% copy %

   (copytx) PROCEDURE (ptr1, ptr2, ptr3, astring);
      %copy word or visible%
      REF ptr1, ptr2, ptr3, astring;
      ST ptr1 ← SF(ptr1) ptr1, *astring*,
         $ptr2 ptr3, %dont move pointers%
         ptr1 SE(ptr1);
      RETURN;
      END.

% delete %

   (dsttx) PROCEDURE (ptr);
      FIND SF(ptr) ↑sptr1 SE(ptr) ↑sptr2;
      <UTILTY, cldsr> ($sptr1);
      delst(ptr);
      RETURN;
      END.

   (deltx) PROCEDURE (ptr1,ptr2);
      REF ptr1, ptr2;
      cldtxt(&ptr1, &ptr2);
      ST ptr1 ← SF(ptr1) ptr1, ptr2 SE(ptr1);
      RETURN;
      END.

   (incspc) PROCEDURE (ptr1,ptr2);
      REF ptr1, ptr2;
      FIND ptr2 > (SP ↑ptr2 / ptr1 < (SP ↑ptr1 / TRUE));
      RETURN;
      END.

% insert %

   (insrtx) PROCEDURE (ptr1, astrng);
      REF ptr1, astrng;
      ST ptr1 ← SF(ptr1) ptr1, *astrng*, ptr1 SE(ptr1);
      RETURN;
      END.

   (insrvs) PROCEDURE (ptr1, astrng);
      REF ptr1, astrng;
      ST ptr1 ← SF(ptr1) ptr1, SP, *astrng*, ptr1 SE(ptr1);
      RETURN;
      END.

% move %

   (movetx) PROCEDURE (ptr1, ptr2, ptr3, ptr4, ptr5, astring);
```

```
      REF ptr1, ptr2, ptr3, ptr4, ptr5, astring;
      IF POS SF(ptr1) = SF(ptr2) THEN
         IF POS ptr1 < ptr2 THEN
            ST ptr1 ← SF(ptr1) ptr1, *astring*, ptr2 ptr3,
               ptr1 ptr4, ptr5 SE(ptr1)
         ELSE
            ST ptr1 ← SF(ptr1) ptr4, ptr5 ptr1,
               *astring*, ptr2 ptr3, ptr1 SE(ptr1)
      ELSE
         BEGIN
         ST ptr1 ← SF(ptr1) ptr1,
            *astring*, ptr2 ptr3, ptr1 SE(ptr1);
         ST ptr2 ← SF(ptr2) ptr4, ptr5 SE(ptr2)
         END;
      RETURN;
      END.

% replace %

   (rp11) PROCEDURE (ptr1, ptr2, astring);
      %replace with contents of astring%
      REF ptr1, ptr2, astring;
      cldtxt(&ptr1, &ptr2);
      ST ptr1 ← SF(ptr1) ptr1, *astring*, ptr2 SE(ptr1);
      RETURN;
      END.

   (rp12) PROCEDURE (ptr1,ptr2,ptr3,ptr4);
      %replace with (ptr3 ptr4)%
      REF ptr1, ptr2, ptr3, ptr4;
      cldtxt(&ptr1, &ptr2);
      ST ptr1 ← SF(ptr1) ptr1, ptr3 ptr4, ptr2 SE(ptr1);
      RETURN;
      END.

% transpose %

   (trantx) PROCEDURE (ptr1, ptr2, ptr3, ptr4);
      REF ptr1, ptr2, ptr3, ptr4;
      cldtxt(&ptr1, &ptr2);
      cldtxt(&ptr3, &ptr4);
      IF POS SF(ptr1) = SF(ptr3) THEN
         IF POS ptr1 < ptr3 THEN
            ST ptr1 ← SF(ptr1) ptr1, ptr3 ptr4, ptr2 ptr3,
               ptr1 ptr2, ptr4 SE(ptr1)
         ELSE
            ST ptr1 ← SF(ptr1) ptr3, ptr1 ptr2, ptr4 ptr1,
               ptr3 ptr4, ptr2 SE(ptr1)
      ELSE
         BEGIN
         *lit* ← ptr1 ptr2;
         ST ptr1 ← SF(ptr1) ptr1, ptr3 ptr4, ptr2 SE(ptr1);
         ST ptr3 ← SF(ptr3) ptr3, *lit*, ptr4 SE(ptr3);
         END;
      RETURN END.
% case shift %
```

```
(cshft) PROCEDURE (ptr1,ptr2);
   REF ptr1, ptr2;
   CASE xsmode OF
      =0: % capital %
         ST ptr1 ← SF(ptr1) ptr1, +ptr1 ptr2, ptr2 SE(ptr1);
      =1: % lower %
         ST ptr1 ← SF(ptr1) ptr1, -ptr1 ptr2, ptr2 SE(ptr1);
      ENDCASE;
   RETURN;
   END.

(cshftall) PROCEDURE (stid);
   CASE xsmode OF
      =0: % capital %
         ST stid ← +SF(stid) SE(stid);
      =1: % lower %
         ST stid ← -SF(stid) SE(stid);
      ENDCASE;
   RETURN;
   END.

% link shift %

(lshft) PROCEDURE (ptr);
   REF ptr;
   FIND > ptr CH ↑p2;
   ST ptr ← SF(ptr) ptr, +ptr p2, p2 SE(ptr);
   RETURN;
   END.

% clist update for deletes %

(cldtxt) PROCEDURE(ptr1, ptr2);
   REF ptr1, ptr2;
   FIND ptr1 ↑sptr1 ptr2 ↑sptr2;
   <UTILTY, cldsr> ($sptr1);
   RETURN;
   END.

% used to build statements %

(setrot) PROCEDURE (ptr, fnstng, dtstng);
   REF ptr, fnstng, dtstng;
   FIND SE(ptr) ↑p2 SF(ptr) > [';] [';] ↑p2;
   ST ptr ← *fnstng*, ", " , *dtstng*, SP,
      *initsr*, " ;" , p2 SE(ptr);
   RETURN;
   END.

(stlit) PROCEDURE (ptr, astring);
   REF astring;
   ST ptr ← *astring*;
   RETURN;
   END.
```

```
% build dummy statement %

    (gdmys) PROCEDURE (ptr);
       REF ptr;
       *lit* ← NULL;
       <AUXCOD, getdat>($lit);
       ST ptr ← ", ", *lit*, SP, *initsr*, " ;";
       RETURN;
       END.

FINISH
°tr3, astring;
    ST ptr1 ← SF(ptr1) ptr1, *astring*,
       $ptr2 ptr3, %dont move pointers%
       ptr1 SE(ptr1);
    RETURN;
    END.

% delete %

    (dsttx) PROCEDURE (ptr);
       FIND SF(ptr) ↑sptr1 SE(ptr) ↑sptr2;
       <UTILTY, cldsr> ($sptr1);
       delst(ptr);
       RETURN;
       END.

    (deltx) PROCEDURE (ptr1,ptr2);
       REF ptr1, ptr2;
       cldtxt(&ptr1, &ptr2);
       ST ptr1 ← SF(ptr1) ptr1, ptr2 SE(ptr1);
       RETURN;
       END.

    (incspc) PROCEDURE (ptr1,ptr2);
       REF ptr1, ptr D
```

USR P6M

```
<NLS>USRPGM.NLS;25, 24-JAN-72 17:37 BLP ;
FILE usrpgm   % L10 <Rel-nls>Usrpgm %

REGISTER r1 = 1, r2 = 2;  REF tda;

(qgp)  PROCEDURE;  % command parser for DNLS Goto user Programs %
   LOCAL pgmfield, inswtch, stid, errs;
   LOCAL TEXT POINTER tp;
   LOCAL STRING pgmnam[50], errstr[100], patstr[500];
   ON SIGNAL
      = upgerr: dismes (2, sysmsg);
      ELSE;
   LOOP
      BEGIN
      DSP(← <Goto Programs);
      CASE input() OF
         = 'c:   % compile a Content analyzer pattern %
            BEGIN
            DSP (<Content analyzer pattern compile);
            *patstr* ← NULL;
            INPUT (BUG tp  CA inswtch ← TRUE; /
               TEXT patstr CA inswtch ← FALSE;);
            IF NOT inswtch THEN
               BEGIN
               *patstr* ← *patstr*, "; ";
               FIND SF(*patstr*) ↑tp;
               END;
            IF (errs ← xgpc ($tp, lda())) THEN
               BEGIN
               *errstr* ← STRING(errs), " error(s):  Type CA.";
               dismes (1, $errstr);
               clrbuf (0);  % clear input buffer %
               LOOP IF input() = CA THEN EXIT;
               dismes (0);
               END;
            IF inswtch THEN
               BEGIN bmoff(); *pgmnam* ← NULL; dn($pgmnam); END
            ELSE rstlit();
            END;
         = 'd:   % "Deinstitute" one of the programs %
            BEGIN
            DSP (<Deinstitute program ↑);
            *pgmnam* ← NULL;
            INPUT WORD pgmnam CA;
            DSP (←);
            xgpd ($pgmnam);
            bmoff(); *pgmnam* ← NULL; dn($pgmnam);
            END;
         = 'e:   % execute one of the programs %
            BEGIN
            DSP (<Execute program ↑);
            *pgmnam* ← NULL;
            INPUT WORD pgmnam CA;
            DSP (←);
            xgpe ($pgmnam);
            bmoff(); *pgmnam* ← NULL; dn($pgmnam);
```

```
    END;
= 'i:   % institute a user program %
    BEGIN
    DSP (<Institute program ↑);
    *pgmnam* ← NULL;
    INPUT WORD pgmnam CA;
    DSP (<as ↑Content analyzer);
    CASE input() OF
       = 'c:
          BEGIN
          DSP (<as Content analyzer   OK);
          IF input() # CA THEN REPEAT CASE (curchr);
          pgmfield ← dacacode;
          END;
       = 'k:
          BEGIN
          DSP (<as Key extractor   OK);
          IF input() # CA THEN REPEAT CASE (curchr);
          pgmfield ← daukeycod;
          END;
       = 's:
          BEGIN
          DSP (<as Sequence generator   OK);
          IF input() # CA THEN REPEAT CASE (curchr);
          pgmfield ← dausqcod;
          END;
       = CA:   REPEAT CASE ('c);
       = CD:   GO TO STATE;
       ENDCASE
          BEGIN
          qm();
          REPEAT CASE;
          END;
    DSP (←);
    xgpi ($pgmnam, lda(), pgmfield);
    bmoff(); *pgmnam* ← NULL; dn($pgmnam);
    END;
= 'l:   % L10 user program compile %
    BEGIN
    DSP (<L10 user program compile);
    INPUT STID stid CA;
    IF (errs ← xgpl (stid, lda())) THEN
       BEGIN
       *errstr* ← STRING(errs), " error(s):  Type CA.";
       dismes (1, $errstr);
       clrbuf (0);  % clear input buffer %
       LOOP IF input() = CA THEN EXIT;
       dismes (0);
       END;
    bmoff();
    END;
= 'p:   % pop program off stack of user programs %
    BEGIN
    DSP (<Pop program from stack   OK);
    wait();
    xgpp();
```

```
            END;
         = 'r:   % reset stack of user programs %
            BEGIN
            DSP (<Reset program stack   OK);
            wait();
            xgpr();
            END;
         = 's:   % status of user programs %
            BEGIN
            DSP (<Status of user programs);
            wait();
            xgps ($lit, dsparea(lcda()));
            *lit* ← *lit*, EOL, EOL, "Type CA";
            <LITDSP, litdpy> ($lit);
            wait();
            rstlit();
            END;
         = BC, = BW:   REPEAT LOOP;
         = CD: GOTO STATE;
         ENDCASE <NLS, NCTRL, wc> ();
      END;
   END.

(tgp)  PROCEDURE;  % command parser for TNLS Goto user Programs %
   LOCAL pgmfield, stid, errs;
   LOCAL TEXT POINTER tp;
   LOCAL STRING pgmnam[50], errstr[100], patstr[500];
   ON SIGNAL
      = upgerr:
         BEGIN
         crlf();
         typeas (sysmsg);
         GOTO STATE;
         END;
      ELSE;
   echo ($"Programs ");
   CASE inpcuc() OF
      = 'C:   % compile a Content analyzer pattern %
         BEGIN
         echo ($"Content analyzer ");
         *patstr* ← NULL;
         prompt ($" type in?");
         CASE inpcuc() OF
            = SP, = CA, = 'Y:
               BEGIN
               prompt ($" Yes");
               crlf();
               txtlit ($patstr);
               *patstr* ← *patstr*, "; ";
               FIND SF(*patstr*) ↑tp;
               END;
            = 'N:
               BEGIN
               prompt ($" No");
               tbug ($tp);
               END;
```

```
        ENDCASE error();
      IF (errs ← xgpc ($tp, &tda)) THEN
        BEGIN
        *errstr* ← EOL, STRING(errs), " error(s):  Type CA.", EOL;
        typeas ($errstr);
        clrbuf (0);
        getctc();
        END;
      END;
    = 'D:   % "Deinstitute" one of the programs %
      BEGIN
      echo ($"Deinstitute program ");
      *pgmnam* ← NULL;
      txtlit ($pgmnam);
      xgpd ($pgmnam);
      END;
    = 'E:   % execute one of the programs %
      BEGIN
      echo ($"Execute program ");
      *pgmnam* ← NULL;
      txtlit ($pgmnam);
      xgpe ($pgmnam);
      END;
    = 'I:   % institute a user program %
      BEGIN
      echo ($"Institute program ");
      *pgmnam* ← NULL;
      txtlit ($pgmnam);
      crlf();
      prompt ($"as ");
      CASE inpcuc() OF
        = 'C:
          BEGIN
          echo ($"Content analyzer ");
          IF inpcuc() # CA THEN REPEAT CASE (curchr);
          pgmfield ← dacacode;
          END;
        = 'K:
          BEGIN
          echo ($"Key extractor ");
          IF inpcuc() # CA THEN REPEAT CASE (curchr);
          pgmfield ← daukeycod;
          END;
        = 'S:
          BEGIN
          echo ($"Sequence generator ");
          IF inpcuc() # CA THEN REPEAT CASE (curchr);
          pgmfield ← dausqcod;
          END;
        = CA:   REPEAT CASE ('C);
        ENDCASE error();
      xgpi ($pgmnam, &tda, pgmfield);
      END;
    = 'L:   % L10 user program compile %
      BEGIN
      echo ($"L10 compile ");
```

```
        prompt ($"at ");
        tbug ($tp);
        IF (errs ← xgpl (tp, &tda)) THEN
           BEGIN
           *errstr* ← EOL, STRING(errs), " error(s):  Type CA.", EOL;
           typeas ($errstr);
           clrbuf (0);  % clear input buffer %
           getctc();
           END;
        END;
     = 'P:   % pop program off stack of user programs %
        BEGIN
        echo ($"Pop stack ");
        getctc();
        xgpp();
        END;
     = 'R:   % reset stack of user programs %
        BEGIN
        echo ($"Reset stack ");
        getctc();
        xgpr();
        END;
     = 'S:   % status of user programs %
        BEGIN
        echo ($"Status type ");
        getctc();
        xgps ($lit, &tda);
        crlf();
        typeas ($lit);
        crlf();
        END;
     ENDCASE error ();
  GOTO STATE
  END.

(gpgmrst)  PROCEDURE;  % Goto User program reset %
  % reset the stack and all display area descriptor fields having to
  do with user programs %

  LOCAL da;
  REF da;
  upgcacnt ← 0;   % number of Content analyzer patterns compiled %
  upgffbuf ← $upgbuf;
     % address of first free cell in buffer used to hold compiled code
     from Goto L1O and Execute Content Analyzer commands. %
  FOR upgskix DOWN UNTIL = 0 DO popsym();
     % reset DDT's symbol table %
  *upgnms* ← NULL;
     % string containing names of user programs -- nth visible in
     string corresponds to nth entry in stack %
  FOR &da ← $dpyare UP dal UNTIL >= $dpyare + dacnt*dal DO
     IF da.daexis THEN da.dacacode ← da.dausqcod ← da.daukeycod ← 0;
  RETURN;
  END.

(gpconan)  PROCEDURE   % does Goto Program Content analyzer %
```

```
    (tpb,  % text pointer to string containing pattern %
    da);       % display area %

    % Compile a content analyzer pattern.  If the compilation is
    successful and there is room in the buffer for the code and room in
    the user program stack, add the pattern to the stack of user
    programs and make it the current CONAN program -- errors cause the
    return of the address of a string containing an error message.
    DDT's symbol table is marked. %

    LOCAL size, errs;
    LOCAL TEXT POINTER tpe;
    REF tpb, da;

    marksym (0);  % mark the symbol table %
    [upgffbuf] ← $upgbend - upgffbuf;   % space left in buffer %
    errs ← <SEQFIL, processor>
       (IF exp THEN $x110nam ELSE $110nam, &tpb, &da, upgtyp, upgffbuf,
       0  :size);
    IF errs > 0 THEN
       BEGIN
       popsym();
       RETURN (errs)
       END
    ELSE IF upgskix >= $upgsksz THEN
       BEGIN
       popsym();
       SIGNAL (upgerr, $"no more room in user program stack")
       END
    ELSE
       % compilation was successful and there was room in the buffer and
       there is room in stack %
       BEGIN
       da.dacacode ← upgstk [upgskix ← upgskix + 1] ← upgffbuf;
       upgffbuf ← upgffbuf + size;
       BUMP upgcacnt;
       FIND tpb $NP ↑tpb $5 PT ↑tpe;
       *upgnms* ← *upgnms*, SP, "UP", STRING(upgcacnt), '↓, tpb tpe;
       END;
    RETURN (0);
    END.

(gpdeinst) PROCEDURE   % does Goto Program Deinstitute a program %
    (str);   % string containing name or number of the user program %
    % "deinstitute" the specified user program --
       find any references to it and delete them, but don't give back
       its space to the buffer -- thus it can re "reinstituted" %
    REF str;
    dinstupg (upgcnv (&str));
    RETURN;
    END.

    (dinstupg) PROCEDURE   %  Deinstitute a program %
       (pgmaddr);   % address of of the user program %
       % "deinstitute" the specified user program --
          find any references to it and delete them, but don't give back
```

```
      its space to the buffer -- thus it can "reinstituted" %
   LOCAL da;
   REF da;
   FOR &da ← $dpyare UP dal UNTIL >= $dpyare + dacnt*dal DO
      IF da.daexis THEN
         BEGIN  % fields pointing to program to be popped are reset
         %
         IF da.dacacode = pgmaddr THEN da.dacacode ← 0;
         IF da.dausqcod = pgmaddr THEN da.dausqcod ← 0;
         IF da.daukeycod = pgmaddr THEN da.daukeycod ← 0;
         END;
   RETURN;
   END.

(gpexpgm)  PROCEDURE   % does Goto Program Execute a program %
   (str);   % string containing name or number of the user program %
   % control is transferred to specified program by means of a CALLO%
   LOCAL upgaddr;
   REF str;
   upgaddr ← upgcnv (&str);
   IF NOT upgaddr > 0 THEN
      SIGNAL (upgerr, $"no address for program");
   [upgaddr]();
   RETURN;
   END.

(gpinst)   PROCEDURE   % does Goto Program Institute a program %
   (str,    % string containing name or number of the user program %
   da,      % display area %
   field);  % field of da in which to institute %
   % make one of the already compiled user programs the content
   analyzer, the user sequence generator, or the user sort key
   extractor program for the specified display area %
   REF str, da;
   da.field ← upgcnv (&str);
   RETURN;
   END.

(gpl1l0)    PROCEDURE   % does Goto Program L1O program compile %
   (stid, % stid where compilation is to start %
   da);   % display area descriptor %

   % Compile a L1O program.  If the compilation is successful and there
   is room in the buffer for the code and room in the user program
   stack, add the program to the stack of user programs, but do not
   make it the current CONAN program.  Mark DDT's symbol table. %

   LOCAL size, errs;
   LOCAL TEXT POINTER tp, tpb, tpe;
   LOCAL STRING str[50];
   REF da;

   IF NOT (FIND SF(stid) ↑tp [("PROGRAM"/"FILE")] $NP ↑tpb $LD ↑tpe)
   THEN
      SIGNAL (upgerr, $"couldn't find program name");
   *str* ← tpb tpe;
```

```
   marksym ($str);
   /upgffbuf/ ← $upgbend - upgffbuf;   % space left in buffer %
   errs ← <SEQFIL, processor>
      (IF exp THEN $xllOnam ELSE $llOnam, &tp, &da, upgtyp, upgffbuf, 0
       :size);
   IF errs > 0 THEN
      BEGIN
      popsym();
      RETURN (errs)
      END
   ELSE IF upgskix >= $upgsksz THEN
      BEGIN
      popsym();
      SIGNAL (upgerr, $"no more room in user program stack")
      END
   ELSE
      % compilation was successful and there was room in the buffer and
      there is room in stack %
      BEGIN
      upgstk /upgskix ← upgskix + 1/ ← upgffbuf;
      upgffbuf ← upgffbuf + size;
      *upgnms* ← *upgnms*, SP, *str*;
      END;
   RETURN (0);
   END.

(gppop)    PROCEDURE;  % does Goto Program Pop a program %
   % Delete top program on user program stack --
      pop DDT's symbol table
      find any references to it and delete them
      give back its space to the buffer
      fix the stack pointer and the string of program names %
   LOCAL TEXT POINTER tp;
   IF upgskix <= 0 THEN SIGNAL (upgerr, $"stack is empty");
   popsym ();
   dinstupg (upgstk/upgskix/);
   IF upgstk/upgskix/ IN /$upgbuf, $upgbend) THEN
      % if program is in buffer, free the buffer space %
      upgffbuf ← upgstk/upgskix/;
   BUMP DOWN upgskix;
   FIND SE(*upgnms*) $PT $NP ↑tp;
   *upgnms* ← SF(*upgnms*) tp;
   RETURN;
   END.

(gpstatus) PROCEDURE   % does Goto Program Status display %
   (str, da);   % address of string in which to put stuff %
   %make up a string containing the status of the user program stuff%
   REF str, da;
   *str* ←
      "Stack of compiled programs (first is #1):", EOL,
         "   ", *upgnms*, EOL, EOL,
      "Room left in buffer:  ",
         STRING ($upgbend - upgffbuf),  " words.", EOL, EOL;
   *str* ← *str*,
      "Content Analyzer  ", " program for display area:  ";
```

```
    upgsstr (da.dacacode, &str);
    *str* ← *str*,
       "Sequence Generator", " program for display area:  ";
    upgsstr (da.dausqcod, &str);
    *str* ← *str*,
       "Sort Key Extractor", " program for display area:  ";
    upgsstr (da.daukeycod, &str);
    RETURN;
    END.

    (upgsstr) PROCEDURE (pgmaddr, str);
       LOCAL i;
       LOCAL TEXT POINTER tp1, tp2;
       LOCAL STRING lstr[50];
       REF str;
       IF pgmaddr = 0 THEN *lstr* ← "None"
       ELSE
          BEGIN
          *lstr* ← "None";
          FOR i ← upgskix DOWN UNTIL = 0 DO
             IF upgstk[i] = pgmaddr THEN
                BEGIN
                FIND SF(*upgnms*);
                FOR i ← i - 1 DOWN UNTIL = 0 DO FIND $NP $PT;
                FIND $NP ↑tp1 $PT ↑tp2;
                *lstr* ← tp1 tp2;
                EXIT LOOP 1;
                END;
          END;
       *str* ← *str*, *lstr*, EOL;
       RETURN;
       END.

(upgcnv)   PROCEDURE   % get user program address from name or number%
    (str);   % address of string containing name or number %
    % given the name or number of a user program in a string, look up
    the name in the string of user program names and/or the address in
    the stack of user program starting addresses %
    LOCAL pgmidx;
    REF str;
    IF str.L = empty THEN pgmidx ← 0
    ELSE IF *str* [1] IN ['0, '9] THEN pgmidx ← cvsno (&str)
    ELSE
       BEGIN
       pgmidx ← upgskix;
       FIND SE(*upgnms*) $NP;
       WHILE NOT (FIND < ENDCHR) DO
          BEGIN
          FIND < $NP $PT;   % move to next visible %
          IF (FIND > *str* (NP/ENDCHR)) THEN EXIT LOOP
          ELSE BUMP DOWN pgmidx;
          END;
       END;
    IF NOT pgmidx IN [1, upgskix] THEN
       SIGNAL (upgerr, $"illegal user program spec");
    RETURN (upgstk[pgmidx]);
```

```
   END.

(popsym)   PROCEDURE;  % pops block of symbols from DDT's symbol table
%
   r1 ← 2;
   r2 ← 0;
   IF NOT SKIP !JSP 0,770002B THEN
      SIGNAL (upgerr, $"DDT symbol table underflow");
   RETURN;
   END.

(marksym)  PROCEDURE   % marks DDT's symbol table %
   (blocknm);   % 0 or string containing the block name %

   REF blocknm;
   IF &blocknm THEN
      BEGIN
      *str* ← *blocknm*, 0;
      r2 ← chbptr (empty) + $str;
      END
   ELSE r2 ← 0;
   r1 ← 1;
   IF NOT SKIP !JSP 0,770002B THEN
      SIGNAL (upgerr, $"DDT symbol table underflow");
   RETURN;
   END.

FINISH
```

UTILITY

```
<NLS>UTILTY.NLS;113, 16-DEC-71 14:04 MSC ; ["err"];
FILE utilty % L10 <rel-nls>utilty %
%Record Definitions%

   (viewspecs) RECORD
      vslev[6],      %lower level bound--level clipping%
      vstrnc[6],     %line truncation value%
      vsrlev[6],     %relative level%
      vslevd[1],     %direction of relative level adjustment%
      vscapf[1],     %content analyzer pass-flag%
      vsusqf[1],     %user sequence generator flag%
      vsbrof[1],     %branch only flag%
      vsplxf[1],     %plex-only flag%
      vsblkf[1],     %blank line flag%
      vsindf[1],     %indenting on flag%
      vsnamf[1],     %names on flag%
      vsstnf[1],     %loc-nums on flag%
      vsstnr[1],     %loc-nums on right flag%
      vsaflf[1],     %abbreviated feedback line flag%
      vsmkrf[1],     %display markers flag%
      vsfrzf[1],     %frozen statements flag%
      vscakf[1],     %content analyzer k viewspec flag%
      vsidtf[1],     %initials, date, and time flag%
      vspagf[1],     %page flag for tty output%
      vsdaft[1];     %display area format flag%

   (seqr) RECORD   % sequence generator work area -- sqwrkl words long %
      swcstid  [36],  % real STID of current statement %
      swlbstid [36],  % STID of statement heading last branch in the
      sequence %
      swstid   [36],  % "stid" of last item "port-sent" from this
      sequence -- it may be an ENDFIL, a stastr (pointer to an
      a-string), or the same as SWCSTID %
      swusqcod [18],  % address of user sequence generator code or 0 %
      swcacode [18],  % address of content anylyzer code or 0 %
      swvspec  [36],  % first word of viewspecs %
      swvsp2   [36],  % second word of viewspecs %
      swsrsav  [36],  % save area for s-register when switch stacks %
      swmrsav  [36],  % save area for m-register when switch stacks %
      swsvw    [18],  % address of statement vector work area %
      swslvl   [6],   % level at which sequence was started %
      swclvl   [6],   % current statement's level %
      swkflg   [1],   % has first item in this sequence been port-send
      -- used for k viewspec %
      swalloc  [1],   % whether or not this work area is allocated %
      swcall   [1];   % is user seqgen (or SSEQGEN) to be CALLed -- or
      gotten to thru the "port-send" mechanism? %
      SET EXTERNAL sqwrkl = 9;

   (ring) RECORD %ringl is length%
      rsub[18],     %psid of sub of this statment%
      rsuc[18],     %psid of suc of this statement%
      rsdb[18],     %psdb of sdb for this statement%
      rinstl[7],   %DEX interpolation string-- scratch space%
      rinst2[7],   %DEX interpolation string-- scratch space%
      rdummy[1],    %DEX dummy flag-- scratch space%
```

```
   repet[3],     %DEX repetition-- scratch space%
   rhf[1],       %head flag. true if this is head of plex%
   rtf[1],       %tail flag, true if tail of plex%
   rnamef[1],    %name flag, true if statement has a name%
   rnull[2],     %unused%
   rnameh[30],   %name hash for this statement%
   rsid[30];     %statement identifier%
   %although only need four words, use five so that have room
   to grow%

(cabts) RECORD  %content analysis bits%
   cackf[1],     %true if have tested with current pattern%
   capsf[1];     %true if have passed with current pattern.%
(ckpbits) RECORD % checkpoint bits %
   ckpt1[1],     % need to update first checkpoint file %
   ckpt2[1],     % need to update second checkpoint file %
   nxckpt[1];    % which checkpoint to update next %
(sdbhead) RECORD %sdbhdl is length%
   sgarb[1],     %true if this sdb is garbage%
   slength[9],   %number of words in this sdb%
   schars[11],   %number of characters in this statement%
   slnmdl[7],    %left name delimiter for statement%
   srnmdl[7],    %right name delimiter for statement%
   spsid[18],    %psid of the statement for this sdb%
   sname[11],    %position of character after name%
   stime[36],    %date and time when this sdb created%
   sinit[21];    %initials of user who created this sdb%
   %sgarb and slength must be in the first word of the header
   for newsdb% %although only need four words, use five so
   that have room to grow%

(fileblockheader) RECORD %fbhdl is length%
   fbnull[36],     %unused%
   fbind[9],     %status table index%
   fbpnum[9],    %page number in file of this block%
   fbtype[5];    %type of this block
      hdtyp   = header
      sdbtyp  = data
      rngtyp  = ring
      jnktyp  = misc (such as keyword, viewchange etc.)%

(corpgr) RECORD %one word.  core page status record.  gives status
for a given core page for random files.%
   ctfull[1],     %true if the page is in use%
   ctfile[4],     %file to which the page belongs%
   ctpnum[9],     %page number within the file%
   ctfroz[3];     %number of reasons why frozen%

      %The array CORPST is the core page status table and is
      made up of instances of the above record.  RFPMAX gives
      the number of core pages that may contain file pages.
      The core pages are located at positions indicated by
      the array CRPGAD (core page address).  CORPST is indexed
      by numbers in the range [1, RFPMAX).  The starting
      location of page k is given by crpgad[k].%
```

```
(rfstr) RECORD % Random file block status record. (=0 then
unallocated, else one of the following records).%

    rfexis[1],   %true if the block exists in the file%
    rfpart[1],   %true if block comes from partial copy%
    rfnull[2],   %unused%
    rfused[10],  %used word count for the block%
    rffree[10],  %free pointer for the block%
    rfcore[9];   %0 then not in core, else page index%

    % The table RFBS is broken into two sections each of which
    contains a block a records of the above type. The first section
    includes RNGM entries from RFBS[RNGBAS] up to and including
    RFBS[RNGBAS+RNGM-1] and contains information about the ring blocks
    in the file. The second section includes DTBM entries from
    RFBS[DTBBAS] up to and including RFBS[DTBBAS+DTBM-1] and contains
    information about the data blocks in the file. The entry
    RFBS[RNGBAS+i] may also be referenced as RNGST[i]; likewise
    RFBS[DTBBAS+i] may be referenced as DTBST[i]. The index in RFBS
    of a block is the actual page number of the block in the file. %

    % Data blocks are allocated in the file starting with page DTBBAS.
     Up to DTBM data blocks may be allocated, with data blocks given
    internal numbers from 0 to DTBM-1. The array DTBST in the file
    header is the data block status table and contains a one word
    record for each potential data block. A zero entry means that the
    block does not exist in the file, otherwise the entry is as
    described in the above record defintion. A pointer to an SDB
    (PSDB) consists of a nine bit data block number in the range
    [0,DTBM) and a nine bit displacement from the start of the block.
    The variable DTBL is maintained in each file header as the current
    upper bound on allocated data blocks for that file. This is used
    to limit the search for a location for a new SDB. The variable
    DTBLST contains the index of the block from which an SDB was last
    allocated or freed. %

(filstr) RECORD %File status table record. entry length = filstl,
max no. entries = filmax%
    flexis[1],   %true: entry represents an existant file%
    flhead[9],   %crgpad index of the file header%
    flbrws[1],   %this file in browse mode%
    fllock[1], %This file was locked by another user when loaded%
    flaccm[8],   %file access mask%
    fldirno[12],  %directory number for the original file%
    flpart[18],  %JFN for the partial copy%
    flbpart[18], %JFN for the browse partial copy%
    florig[18],  %JFN for the original file%
    flastr[18],  %address of the file name string%
    flpcst[18],  %address of partial copy name string%
    flbpcst[18]; %address of browse partial copy name string%
    DECLARE EXTERNAL filstl = 4, filmax = 15;

(stidr) RECORD stpsid[18], stfile[4], stastr[1];
    % -- (filmnp, lodrng) and (filmnp, lodsdb) cheat and make use of
    the format of an stid -- so please don't change this without
    checking those routines and telling WHP %
```

```
   (stbw) RECORD stwc[9], stblk[9];
   (stsdb) RECORD stpsdb[18];
   (stast) RECORD stadr[18];
   (stsidr) RECORD stsidf[4], stsid[30];

(jsent) RECORD %jump stack entry -- 3 words%
   jsstid[36], %stid, this entry%
   jsvsp1[36], %viewspecs, this entry (first word)%
   jsvsp2[36]; %viewspecs, this entry (second word)%

(jshdr) RECORD %jump stack header -- 3 words%
   jsbotm[18],  %bottom of jump stack%
   jscurr[18],  %current element in jump stack%
   jsnext[18],  %location of next jump stack header%
   jslink[18],  %location of link entry for this stack%
   jsage[6]; %age of this jump stack%

(lsent) RECORD %link stack entry --  3 words%
   lfilnm[18], %address A-string file name for this entry%
   ljshdr[18], %address of jump stack header%
   lsnext[18], %address of next link stack entry%
   lsprev[18], %address of previous link stack entry%
   lsage[8], %age of this entry%
   ljsmult[1],
      %jump stack flag; if TRUE, jump stack allocated for this
      entry%
   lsnew[1]; %if TRUE, no jump stack entries for this file%

(clistr) RECORD %clist entry%
   clst1[36],  %original or updated stid%
   clst2[36],  %successor stid%
   clcc1[12],  %character count for first%
   clcc2[12],  %character count for second%
   clfixed[1]; %for aptstr%

   DECLARE EXTERNAL cll = 3, clmax = 50;

(clhdr) RECORD %clist header%
   clcnt[9],   %count of entries in use%
   clfno1[7],  %type used to generate clist%
   clfno2[7],  %type used to generate clist%
   cltype[6],  %type used to generate clist%
   clbuff[18]; %address of buffer%

(isqfwa) RECORD %length is 3 words%
   isstid[36], %last completed stid%
   isbuff[18], %sequential file buffer address%
   islevs[18],  %location of string for levadj%
   isinfno[8], %sequential file number%
   istatnd[8], %character that signals end of statement%
   islevb[8], %number of leading blanks per level%
   islstb[8], %number of leading blanks, last statement%
   isdpth[8], %depth from initial psid%
   isfiltyp[1]; %TRUE if tenex file, else from 940%

(lsrt) RECORD  %line segment reference table entry%
```

```
   rtbps[36],
   rtbpe[36],
   rtfcol[18], rtlcol[18],
   rtstid[36],
   rtccnt[12], rtsrce[6], rthinc[18],
   rtx1[18], rtx2[18],
   rty[18], rtlsid[18],
   rtfont[9], rtcsize[2], rtexis[1], rtnew[1], rtllsg[1];

   DECLARE EXTERNAL lsrtl = 8;

(inrptrs) RECORD %pointers to input element%
   inpcur[18],  %last read by lookc%
   inpptr[18];  %last read by input%

(inpsent) RECORD %format of input stack element%
   ipstid[36],  %stid, if marker, else endfil%
   ipchar[36],  %last character read, (char count, if marker)%
   ipcoords[36],  %coordinates of last character read%
   iptime[36];  %time charactr read%

   DECLARE EXTERNAL inpsel = 4;  %length of element on input stack%

(command) RECORD
   copcode[8], cparm1[18], cparm2[18], cda[18];

(frozen) RECORD
   fzvspec[36], fzvspc2[36], fzstid[36], fznext[18], fzexis[1];

   DECLARE EXTERNAL frzl = 4;

(marker) RECORD
   mkname[36], mkpsid[18], mkfix[1], mkccnt[12], mkexis[1];
   DECLARE EXTERNAL mkrl = 2;

(byteptr) RECORD
   bpadr[18], bpindx[6], bpsize[6], bpbitpos[6];

(chars) RECORD
   chnul[1], chr4[7], chr3[7], chr2[7], chr1[7], chr0[7];

(halfword) RECORD
   rhword[18], %right half word%
   lhword[18]; %left half word%

(cords) RECORD
   xcord[18],  %x-cordinate for cursor word%
   ycord[18];  %y-cordinate for cursor word%

(displayarea) RECORD %entrylength = dal, max = damax%
   davspec[36],  %first viewspec word--1st word in record%
   davspc2[36],  %second viewspec word--2nd word in record%
   dacsp[36],    %csp for this da%
   daccnt[12],   %character count for dacsp t-pointer%
   dacsize[2],   %character size%
   danocs[2],    %horizontal increment%
```

```
    dasgcs[2],     %horizontal increment%
    dafrz1[18],    %A(frozen statement chain for this da)%
    daind[18],     %indentation per level%
    damind[18],    %max indentation%
    dacrow[18],    %current row count%
    damrow[18],    %max row count%
    daccol[18],    %current column count%
    damcol[18],    %max column count%
    dahinc[18],    %horizontal increment%
    davinc[18],    %vertical increment%
    danohi[18],    %horizontal increment%
    dasghi[18],    %horizontal increment%
    dafont[18],    %default font%
    dalink[18],    %A(link-jump stack for this da)%
    dahandle[18], %system handle for this display area%
    daleft[18],    %left boundry of da (raster units)%
    daright[18],   %right boundry of da (raster units)%
    datop[18],     %top boundry of da (raster units)%
    dabottom[18], %bottom boundry of da (raster units)%
       %(0,0) is upper left corner%
    darts[9],      %LSRT entry number for start of this da%
    darte[9],      %LSRT entry number for start of next da%
    daempty[1],    %da-has-no-display flag%
    dadsp[1],      %update display for this da%
    daexis[1],     %this da entry is in use%
    datab0[18],    %first tab position%
    datab1[18],    %second tab position%
    datab2[18],    %third tab position%
    datab3[18],    %fourth tab position%
    datab4[18],    %fifth tab position%
    datab5[18],    %sixth tab position%
    datab6[18],    %seventh tab position%
    datab7[18],    %eighth tab position%
    datab8[18],    %nineth tab position%
    datab9[18],    %tenth tab position%
    dacacode [18],    % address of content analyzer program -- or 0 %
    dausqcod [18],    % address of sequence generator program -- or 0 %
    daukeycod[18];    % address of sort key extractor program -- or 0 %
    DECLARE EXTERNAL dal = 20, damax = 8;

(card) RECORD  % substitute candidate record %
    catnc[18], %length of test string%
    carnc[18], %length of replacement string%
    catbp[36], %byte ptr to test string%
    carbp[36], %byte ptr to replacement string%
    canxt[18]; %next card for this initial char%
DECLARE EXTERNAL lcard = 4; %length of card in words%

(shed) RECORD  % substitute header record %
    sbrp[36], %pointer to next space for card%
    sbdp[36], %pointer to dispatcher%
    sbas[36]; %pointer to A-string for strings%
DECLARE EXTERNAL lshed = 3; %length of shed in words%
                          4

(measrec) RECORD
    msbegin[36],  %location to begin measurement%
```

sbtt[36],

```
   msend[36],  %location to end measurements%
   msbinstn[36], %instruction displaced to begin measurement%
   mseinstn[36], %instruction displaced to end measurement%
   msmax[36], %maximum number of passes to measure%
   mscurr[36], %current number of passes measured%
   mslclk[36], %clock at beginning of interval%
   mstclk[36]; %total clock time, this series%

   DECLARE EXTERNAL msentl=8;

(adarl) RECORD    %allocate display area record for r1%
   adauly[10],    %upper left y-cordinate%
   adaulx[10],    %upper left x- cordinate%
   adasize[6],    %max no. of strings allowed in da%
   adadev[6];     %user device%

(adar2) RECORD    %allocate display area record for r2%
   adalry[10],    %lower right y-cordinate%
   adalrx[10],    %lower right x- cordinate%
   adadcs[2],     %default character size%
   adadhi[6],     %default horizontal increment%
   adadf[5];      %default font%

(strdal) RECORD   %string manipulation record for r1%
   sstrid[18],    %string identirier%
   sdaid[18];     %display area identifier%

(strda4) RECORD   %string manipulation record for r4%
   sycord[10],    %y-cordinate%
   sxcord[10],    %x- cordinate%
   scsize[3],     %character size%
   shinc[7],      %horizontal increment%
   sfont[6];      %font%

(scsrr) RECORD    %set cursor record for r3%
   sccsiz[2],     %character size%
   schinc[6],     %horizontal increment%
   scfont[5];     %font%

(xc) RECORD    %xcord record for remote displays%
   xc2[6],     %low order 5 bits%
   xc1[6];     %high order five bits%

(yc) RECORD    %ycord record for remote displays%
   yc2[6],     %low order 5 bits%
   yc1[6];     %high order five bits%

(iptmchr) RECORD %time for remote display%
   tmchr6[6], %low order 6 bits%
   tmchr5[6],
   tmchr4[6],
   tmchr3[6],
   tmchr2[6],
   tmchr1[6]; %high order 6 bits%
(substr) RECORD %substitute stack entry record%
   sbc1cnt [12], sbc2cnt [12], sbc3cnt [12];
```

```
    (deliverymodes) RECORD %Record containing flags for Journal delivery%
       delol[1], %Deliver on-line%
       delhc[1]; %Deliver hard copy%
    (tenexbits)RECORD %Bits according to the TENEX way%
       bitfiller[32],
       bit3[1],
       bit2[1],
       bit1[1],
       b0[1];
%Declarations%

    REGISTER
       r1=1, r2=2, r3=3, r4=4, r5=5, p=7, wa=8, s=9, m=10, rp=11,
       a1=12, a2=13, a3=14, a4=15;
    DECLARE EXTERNAL FIELD  %formats for fields returned by wsi%
       kbdchr = [(rp), 8 : 0],  %keyboard character code%
       kstchr = [(rp), 5 : 8],  %keyset character code%
       inpcse = [(rp), 3 : 16],  %mouse case shift%
       inpsrc = [(rp), 5 : 20],  %input source%
       inpbut = [(rp), 3 : 22],  %button source%
       inpxcd = [(rp), 10 : 0],  %x-coordinates%
       inpycd =  [(rp), 10 : 12],  %y-coordinates%
       btnste = [(rp), 3 : 0],  %state of mouse buttons%
       btncng = [(rp), 3 : 3];  %change in buttons%

    DECLARE EXTERNAL FIELD  %fields used to save sdb initials%
       cint4 =[(rp), 5:0],
       cint3 =[(rp), 5:5],
       cint2 =[(rp), 5:10],
       cint1 =[(rp), 5:15],
       oldint = [(rp), 21:15];
    SET EXTERNAL %for all JSYS's used in NLS%
       login = 1B,
       lgout = 3B,
       getab = 10B,
       sysgt = 16B,
       erstr=11B,
       geter=12B,
       gjinf = 13B,
       time=14B,
       runtm=15B,
       gtjfn=20B,
       openf=21B,
       closf=22B,
       rljfn=23B,
       gtsts=24B,
       delf = 26B,
       sfptr=27B,
       jfns=30B,
       sizef = 36B,
       gactf=37B,
       stdir=40B,
       dirst=41B,
       cndir = 44B,
       bin=50B,
```

```
bout=51B,
sin=52B,
sout=53B,
pmap=56B,
rpacs=57B,
spacs=60B,
gtfdb=63B,
chfdb = 64B,
pbin=73B,
pbout=74B,
psout=76B,
cfibf=100B,
cfobf=101B,
dobe=104B,
gtabs=105B,
stabs=106B,
rfmod=107B,
sfmod=110B,
rfcoc=112B,
sfcoc=113B,
dtach = 115B,
sir=125B,
eir=126B,
aic=131B,
iic = 132B,
dic = 133B,
rcm = 134B,
debrk=136B,
ati=137B,
dti = 140B,
rpcap = 150B,
epcap = 151B,
cfork = 152B,
dismS=167B,
haltf=170B,
gtrpw=171B,
get=200B,
sevec=204B,
gpjfn=206B,
spjfn=207B,
setnm = 210B,
odtim=220B,
idtim=221B,
odcnv = 222B,
nout=224B,
nin=225B,
gtad=227B,
wsiz = 414B,
gtpsw = 415B,
jobtm = 424B,
rdclk = 426B,
delnf = 430B,
flgin = 431B,
tstfg = 432B,
setfg = 433B,
rstfg = 434B,
```

dtchtac = 416 B

```
        wsi = 435B,
        clwsb = 436B,
        ada = 440B,
        dda = 441B,
        strda = 442B,
        cbda = 443B,
        sdda = 444B,
        ssda = 445B,
        rdda = 446B,
        rsda = 447B,
        scsr = 450B,
        tsnda = 451B,
        tsfda = 452B,
        rstda = 453B,
        wsdbs = 454B,
        nlscr = 522B;
    SET %for opcode's%
        JSYS=104B, ADD=270B, SUB=274B, ADDI=271B, AOBJN=253B,
        AOJ=340B, BLT=251B, MOVE=200B, MOVN=210B, PUSH=261B,
        POP=262B, JRST=254B, LDB=135B, ILDB=134B, LSHC=246B,
        HRR=540B, HRRZ=550B, HRL=504B;
    DECLARE EXTERNAL
        empty = 0, endfil = -1,
        origin = 2 %=fbhdl%, nullch = 0, chbmty = 4407000000001B;
    DECLARE %pop transfer table% poptab=(0,
        %1%  qcall, %no argument call pop%
        %2%  qcalll, %single argument call pop%
        %3%  qcallm,  %up to 13 argument call pop%
        %4% qcallm2); %More than 13 argument call pop%

%call and pop routines%

    (sysrtn) PROCEDURE; %for returns%
        %get here by a JRST sysrtn%
        !MOVE s,m;  %back up the stack%
        !POP s,a4;  %save the return location%
        !POP s,m;  %restore the mark%
        !JRST (a4) END.

    (sys2rt) PROCEDURE; %return with 2 results%
        !MOVE s,m;
        !POP s,a4;
        !POP s,m;
        !MOVEM a2,3(s);
        !JRST (a4) END.

    (sysmrt) PROCEDURE; %for returns with multiple results%
        % register a2 contains number-1 of multiple results%
        !MOVN a3,a2;
        !ADD a3,s;
        !HRL a3,a3; %a3 left points to first on stack%
        !HRR a3,m;
        !AOJ a3,; %a3 right points to destination%
        !HRRZ a4,a3;
        !ADD a4,a2; %a4 points to last loc moved to%
        !BLT a3,@a4;
```

```
   !JRST sysrtn END.

(syssig)PROC;
   WHILE [[m-1]].lh = 0 DO
      IF (m .A 18M) <= ([m-1] .A 18M)  THEN
         GOTO stkovr
      ELSE
         IF  [m←[m-1]] = $uflow THEN
            deferr();
   %by here, m points to mark beyond one with signal loc%
   [m] ← [[m-1]].lh; %change return loc to signal code%
   RETURN;
   END.

(repsig)PROC(value);
   %Repeat a signal with value as sysgnl%
   SIGNAL(value);
   END.
(callsig)PROC(value, message);
   %Call signal%
   SIGNAL(value, message);
   END.
DECLARE FIELD popldb=[40B, 9:27], 140ac=[40B, 4:23];
(cllp) PROCEDURE; %dummy procedure for calls%
   (ppaprt):
      %get here by a JSP r5,ppaprt in location 41B%
      IF (r4 ← .popldb) <= 4 %call's% THEN
         BEGIN
         !HRLI r5,0;
         !JRST @poptab(r4);
         END;
      r1 ← $"Illegal UUO at " + 1;
      !HRLI r1,7B2;
      !JSYS psout;
      r1 ← 101B;
      r2 ← (r5 - 1) .A 18M;
      r3 ← 8;
      !JSYS nout;
      !JSYS haltf;
      !JSYS haltf;
   (qcall):
      %get here by a CALL pop%
      !PUSH s,m;  %save the mark%
      !PUSH s,r5;  %save the return location%
      !MOVE m,s;  %new mark%
      !JRST @40B;
   (qcalll):
      %get here by a CALL1 pop%
      !PUSH s,m;  %save the mark%
      !PUSH s,r5; %save the return location%
      !MOVE m,s;  %new mark%
      !MOVEM a1,1(m); %store the single argument%
      !JRST @40B;
   (qcallm):
      %get here by a CALLM pop%
      r4 ← .140ac;
```

```
        !HRL r4,r4;
        !SUB s,r4; %now same as before started call%
        !PUSH s,m;  %save the mark%
        !PUSH s,r5; %save the return location%
        !MOVE m,s;  %new mark%
        !JRST @40B;
    (qcallm2):
        %get here by a CALLM pop%
        r4 ← .140ac;
        !SUB s,@r4; %now same as before started call%
        !PUSH s,m;  %save the mark%
        !PUSH s,r5; %save the return location%
        !MOVE m,s;  %new mark%
        !JRST @40B;
    END.

EXTERNAL ppaprt, qcall, qcalll, qcallm;

%String Construction%

    (bsc) %Begin string construction.  Called with STID (or pointer to
    string with stastr TRUE) of the destination.%

        PROCEDURE (dest);
        rplsid ← dest;
        %set up byte pointer and SAR for string construction%
        nsdbpt ← chbmty + $sar;
        sar.L ← empty;
        %reset stack for modified clist entries%
        clchng ← clmpty;
        RETURN END.

    (esc) %End string construction.%

        PROCEDURE;
        LOCAL
            cnt,    %character count in SAR%
            words,  %number of words used in new string%
            sdbsiz, %size of new SDB%
            stdb,   %stdb for new SDB%
            sdb,    %address of the new SDB%
            sdbpt,  %address of the new SDB%
            end,    %end of SDB header%
            dlleft, %left name delimiter%
            dlrght, %right name delimiter%
            blknum, %core block index for SDB file block%
            rngb,   %ring file block index%
            fhdloc, %address of the file header%
            rnl,    %address of the ring element for the statement%
            cl;     %pointer into clist%
        LOCAL STRING escname[40];
        REF rnl, sdb;
        IF rplsid.stastr THEN %A-string%
            BEGIN
            *[rplsid.stadr]* ← *sar*;
            RETURN;
```

```
    END;
nsdbpt ← chbptr(sar.L) + $sar;
IF sar.L = empty THEN apachr(SP);
cnt ← sar.L;
rngb ← lodrng(rplsid : &rnl);
frzblk(rngb, 1);
stdb ← 0;
IF (stdb.stpsdb ← rnl.rsdb) # 0 THEN
    BEGIN %get the delimiters for the statement%
    stdb.stfile ← rplsid.stfile;
    lodsdb(stdb : &sdb);
    dlleft ← sdb.slnmdl;
    dlrght ← sdb.srnmdl;
    fresdb(rplsid); %get rid of the old sdb%
    END
ELSE IF rplsid.stpsid = origin THEN
    BEGIN %use standard delimiters for that file%
    fhdloc ← filehead[rplsid.stfile] - $filhed;
    dlleft ← [fhdloc + $namdl1];
    dlrght ← [fhdloc + $namdl2];
    END
ELSE
    BEGIN %use same delimiters as successor%
    fchsdb(getsuc(rplsid) : &sdb);
    dlleft ← sdb.slnmdl;
    dlrght ← sdb.srnmdl;
    END;
%find a place for the new SDB%
sdbsiz ← (words ← (sar.L+4)/5) + sdbhdl;
stdb ← newsdb(sdbsiz, rplsid.stfile);
blknum ← lodsdb(stdb : &sdb);
frzblk (blknum, 1);
mvbfbf ($sar+1, &sdb+sdbhdl, words);
%initialize the SDB header%
end ← (sdbpt ← &sdb) + sdbhdl;
DO [sdbpt] ← 0 UNTIL (sdbpt ← sdbpt+1) = end;
sdb.sgarb ← FALSE;
sdb.slength ← sdbsiz;
sdb.slnmdl ← dlleft;
sdb.srnmdl ← dlrght;
sdb.spsid ← rplsid.stpsid;
sdb.schars ← cnt;
sdb.sinit ← cinit;
!JSYS gtad; %get time and date%
sdb.stime ← r1;
%update ring before call to xtrnam%
rnl.rsdb ← stdb.stpsdb;
%check for name%
stcwrk ← rplsid;
stcwrl ← 1;
fechcl (forward, $stcwrk);
xtrnam ($escname, $stcwrk, dlleft, dlrght);
sdb.sname ← IF escname.L = empty THEN 1 ELSE stcwrl;
IF escname.L = empty THEN
    BEGIN
    rnl.rnameh ← 0;
```

```
      rnl.rnamef ← FALSE;
      END
   ELSE
      BEGIN
      astruc($escname);
      rnl.rnameh ← hash($escname);
      rnl.rnamef ← TRUE;
      END;
   %now release the sdb and ring blocks%
   corpst[blknum].ctfroz ← corpst[blknum].ctfroz - 1;
   corpst[rngb].ctfroz ← corpst[rngb].ctfroz - 1;
   UNTIL clchng = clmpty DO
      BEGIN
      POP clchng TO cl;
      [cl].clfixed ← FALSE;
      END;
   RETURN END.
```

(kps) %argument points to string to be appended to the statement being constructed%

```
   PROCEDURE (asfrom);
   REF asfrom;
   LOCAL bptr; %byte pointer for reading characters%
   IF (sar.L ← sar.L + asfrom.L) > sar.M THEN
      err($"Exceed capacity");
   bptr ← chbmty + &asfrom;
   a2 ← asfrom.L;
   a3 ← nsdbpt;
   a4 ← bptr;
   UNTIL (a2 ← a2-1) < 0 DO ↑a3 ← a1 ← ↑a4;
   nsdbpt ← a3;
   RETURN END.
```

(apachr) %passed character as argument. appends to the statement being constructed in sar.%

```
   PROCEDURE (ch);
   IF (sar.L ← sar.L+1) > sar.M THEN err($"Exceed capacity");
   ↑nsdbpt ← ch;
   RETURN END.
```

(aptstr) %Append tstring. Argument is pointer to two T-pointers which delimit the substring to be appended to the statement currently being constructed.%

```
   PROCEDURE (tp);
   LOCAL
      cl,      %location of record for a cl to be updated%
      end,     %location of end of cl table%
      cltab,   %location of cl table%
      flhd,    %file header loc%
      ch,      %character read from the tstring%
      bfrom,    %source byte pointer%
      lenstat,  %number of chars from string%
      lenblank; %number of chars from string%
```

```
REF cl;
IF [tp] # [tp+2] OR [tp+1] <= empty THEN
   err($"illegal string designation");
IF [tp+1] >= [tp+3] THEN RETURN;
stcwrk ← [tp];
stcwrl ← [tp+1];
fechcl (forward, $stcwrk);
% stcwrk[2] now contains length + 1 of source string %
% calculate number of chars to take from source string %
CASE stcwrk[2] OF
   <= [tp+1] :
      BEGIN
      lenstat ← 0;
      lenblank ← [tp+3] - [tp+1];
      END;
   < [tp+3] :
      BEGIN
      lenstat ← stcwrk[2] - [tp+1];
      lenblank ← [tp+3] - stcwrk[2];
      END;
   ENDCASE
      BEGIN
      lenstat ← [tp+3] - [tp+1];
      lenblank ← 0;
      END;
&cl ← cltab ← [clstad].clbuff;
   %address of start of clist%
end ← &cl + [clstad].clcnt * cll;
   %end of clist%
IF modeset THEN
   BEGIN % mode set now in operation %
   UNTIL (lenstat ← lenstat-1) < 0 DO
      BEGIN
      ch ← READC($stcwrk);
      %now append the character%
      CASE modeshift OF %check for forced case%
         =0: apachr(IF ch IN ['a,'z] THEN ch-40B ELSE ch);
            %forced upper case%
         =1: apachr(IF ch IN ['A,'Z] THEN ch+40B ELSE ch);
            %forced lower case%
         ENDCASE apachr(ch); %no case change%
      END;
   modeset ← 0; %reset the global flag%
   END
ELSE %no mode change. just copy across%
   BEGIN
   IF NOT mkrstay THEN
      IF NOT rplsid.stastr AND
         rplsid.stfile = [tp].stfile THEN
         UNTIL &cl = end DO
            BEGIN
            IF cl.clstl = [tp] AND
               cl.clccl IN [[tp+1],[tp+3]) AND
               NOT cl.clfixed THEN
               BEGIN
               cl.clfixed ← TRUE;
```

```
                    cl.clcc1 ←
                       sar.L + 1 + cl.clcc1 - [tp+1];
                    PUSH &cl ON clchng;
                    END;
                 &cl ← &cl + cll;
                 END
           ELSE cldsr(tp);
        IF (sar.L ← sar.L + lenstat) > sar.M THEN
           err($"Exceed capacity");
        a4 ← stcwrk[4];  %byte pointer made by fechc1%
        a2 ← lenstat;
        a3 ← nsdbpt;
        UNTIL (a2 ← a2-1) < 0 DO ↑a3 ← a1 ← ↑a4;
        nsdbpt ← a3;
        END;
     UNTIL (lenblank ← lenblank-1) < 0 DO apachr(SP);
     mkrstay ← FALSE;
     RETURN END.

  (cldsr) %Called with address of two tpointers.  For all clist
  entries, if between these two pointers, then set to "deleted".%

     PROCEDURE (tp);
     LOCAL end, cl;
     REF cl;
     &cl ← [clstad].clbuff;
        %address of start of clist%
     end ← &cl + [clstad].clcnt * cll;
        %end of clist%
     UNTIL &cl >= end DO
        BEGIN
        IF cl.clstl = [tp] AND
           cl.clcc1 IN [[tp+1],[tp+3]) AND
           NOT cl.clfixed THEN
           BEGIN
           cl.clst2 ← cl.clstl := endfil;
           cl.clcc2 ← [tp+1];
           END;
        &cl ← &cl + cll;
        END;
     RETURN END.

%Statement Destruction%

  (delst) %delete the SDB corresponding to the stid passed as argument%

     PROCEDURE (stid);
     fresdb(stid);
     stosdb(stid, 0);
     RETURN END.

%Reading characters from SDB%

  (fechcl)
     %to provide for bounded search:  where now call fechcl, call a new
     procedure opensr, which will initialize PS and PE and then call
```

```
fechc1.  fechc1 will always use the current values of PS and PE to
set up bounds in the work area%
%Documentation%
   %This routine is called to initialize a work area for reading
   characters from a statement.  Arguments are: direction of
   reading characters (=0 then backwards) and the address of the 7
   word work area.
   If characters are to be read from a statement then
      when calling FECHC1, the first two cells of the work area
      must contain a Tpointer.  A character count of one indicates
      the first character of the statement. FECHC1 will initialize
      the rest of the work area. The first word of the word area
      always has the PSID of the statement.  The second word has
      the character count.  The third word contains a bound on
      characters to be read.  ENDCHR's are returned after reach
      this bound.  The fourth word has the direction of readout
      for use by readc.  A READC(x) actually results in the value
      of x being loaded into register wa followed by a JSP
      a4,readc.
      The fifth word of the work area contains a byte pointer to
      the character last read from the statement. Thus an ILDB
      instruction may be used to get the next character if the
      direction is forward.  If the direction of reading is
      backward. then the fifth word contains the address of the
      last character and sixth word contains the position in the
      word of the last character (0 = leftmost).  In this case a
      byte pointer is constructed for each character using the
      information in words five and six.
      To read characters from the statement execute a READC(x)
      where the value of x is the address of the work area to be
      used.  The character is returned as the value of the READC.
      Subsequent READC's will return the following characters.
      To change position or direction within the statement the
      work area must be reinitialized by calling FECHC1 again, as
      described above.  There may however be more than one work
      area currently in use, and these may be changed
      independently.
   If characters are to be read from an A-string then
      the first word of the work area contains the address of the
      A-string instead of a PSID.  The second word is 1 if the
      first character of the string is to be read next, two if the
      second, etc.  Characters may be read out of an A-string in
      either direction, just like a statment.  Endcharacters are
      returned when the string is exhausted.%
PROCEDURE (dir, worka);
LOCAL
   wp,   %word postion of the starting character%
   cp,   %character postion of the starting character%
   stdb,  %stdb for statement%
   rng,  %location of ring element%
   addr; %address of the word containing starting character%
REF rng;
%set work+2 to bound%
IF [worka] = endfil THEN %set to null string%
   BEGIN
   [worka+3] ← 2; %readc will return ENDCHR%
```

```
      RETURN;
      END
   ELSE IF [worka].stastr THEN %A-string%
      BEGIN
      addr ← [worka].stadr;
      [worka+2] ← IF pe THEN pe ELSE [addr].L + 1;
      addr ← addr + 1;
      END
   ELSE %SDB%
      BEGIN
      lodrng([worka] : &rng);
      stdb ← 0;
      stdb.stfile ← [worka].stfile;
      stdb.stpsdb ← rng.rsdb;
      lodsdb(stdb : addr);
      [worka+2] ← IF pe THEN pe ELSE [addr].schars + 1;
      addr ← addr + sdbhdl; %addr of char%
      END;
   %find word and character position%
   DIV ([worka+1]-1)/5, wp, cp;
   IF dir THEN %scan forward%
      BEGIN
      [worka+3] ← 1; %direction%
      %make byte pointer to the previous character%
      cp ← 36 - cp * 7;
      a1 ← addr + wp; !LSHC a1, -24;
      a1 ← 7; !LSHC a1, -6;
      a1 ← cp; !LSHC a1, -6;
      a1 ← worka;
      !MOVEM a2,4(a1);
      END
   ELSE %scan backwards%
      BEGIN
      [worka+2] ← IF ps THEN ps ELSE 1; %change bound%
      [worka+3] ← 0; %direction%
      [worka+4] ← wp + addr;
      [worka+5] ← cp
      END;
   RETURN END.

(openswork) PROCEDURE;
   fechcl(1, $swork);
   RETURN END.

(savpos) PROCEDURE;
   PUSH ps ON btwstk;
   PUSH pe ON btwstk;
   RETURN END.

(respos) PROCEDURE;
   POP btwstk TO pe;
   POP btwstk TO ps;
   RETURN END.

(readc) %This is the routine that is called to read a character by
the READC construct.  Code to call is JSP a4,readc so return loc is
```

in A4.%

```
    PROCEDURE;
    %cant have locals since is not called by usual procedure linkage.
    wa is a register containing the address of the work area.%
    [wa+6] ← a4; %save return loc%
    CASE [wa+3] OF
       =1: %forward scan%
          BEGIN
          IF [wa+2] <= [wa+1] THEN
             BEGIN %have reached the bound%
             [wa+3] ← 2;
             a1 ← ENDCHR;
             END
          ELSE BEGIN
             BUMP [wa+1];
             !ILDB a1,4(wa); %get the character%
             END;
          !JRST @6(wa);
          END;
       =0: %else backward scan%
          BEGIN
          IF [wa+1] <= [wa+2] THEN
             BEGIN %have reached the bound%
             [wa+3] ← 2;
             a1 ← ENDCHR;
             END
          ELSE
             BEGIN
             BUMP DOWN [wa+1];
             IF ([wa+5] ← [wa+5]-1) < 0 THEN
                BEGIN
                [wa+5] ← 4;
                BUMP DOWN [wa+4];
                END;
             a1 ← mkbptr(29-[wa+5]*7, 7, [wa+4]);
             !LDB a1,a1;
             END;
          !JRST @6(wa);
          END;
       ENDCASE %out of bounds%
          BEGIN
          a1 ← ENDCHR;
          !JRST @6(wa);
          END;
    END.
```

(xtrnam) %Arguments are 1: address of A-string, 2: address of work area for READC, 3: left name delimiter, 4: right name delimiter. If instead of passing the delimiters you want xtrnam to use the delimiters given in the SDB header, pass a minus one as the third argument. If finds a name in the statement it is placed in the A-string, otherwise the A-string is set to empty. Uses (TXTEDT, nmdr) to determine the name.%

```
    PROCEDURE (ast, worka, dlleft, dlrght);
```

```
    LOCAL ch, sdb;
    LOCAL TEXT POINTER tp1, tp2, tp3;
    LOCAL STRING dlstr[1], drstr[1];
    REF ast, sdb;
    IF dlleft = -1 THEN %must look up the delimiters%
       BEGIN
       fchsdb([worka] : &sdb);
       dlleft ← sdb.slnmdl;
       dlrght ← sdb.srnmdl;
       END;
    IF dlleft # 0 THEN *dlstr* ← dlleft;
    IF dlrght # 0 THEN *drstr* ← dlrght;
    tp1 ← [worka]; tp1[1] ← [worka+1];
    *ast* ← NULL; %clear a-string to empty%
    FIND tp1 >;
    IF dlleft = 0 OR
    (dlleft = SP AND FIND 1$SP ↑tp1) OR
    FIND $SP *dlstr* $SP ↑tp1 THEN
       BEGIN
       <TXTEDT, nmdr>($tp1, $tp2, $tp3);
       % if no name then tp2 = tp3 %
       FIND tp3 > ↑tp1;
       IF dlrght = 0 OR
       (dlrght = SP AND FIND SP ↑tp1) OR
       FIND $SP *drstr* ↑tp1 THEN
          *ast* ← + tp2 tp3;
       END;
    [worka] ← tp1; [worka+1] ← tp1[1];
    fechcl(1, worka);
    RETURN END.

%A-string routines%

    (asrref) %create a pointer to astring with stastr field set.%

       PROCEDURE (ast);
       ast.stastr ← TRUE;
       RETURN (ast) END.

    DECLARE % byte pointers for chbptr %
       chparray=(
          350700000001B,
          260700000001B,
          170700000001B,
          100700000001B,
          070000000001B);

    (chbptr) %character byte pointer%
       PROCEDURE (charno);
       %creates a partial byte pointer to the character determined by the
       character number passed as argument.  The resulting byte pointer
       needs to have the address of the string added to it.%
       LOCAL wp, cp;
       IF charno = empty THEN RETURN (440700000001B);
       DIV (charno-1) / 5, wp, cp;
       RETURN (chparray[cp] + wp) END.
```

```
(stbptr) %character byte pointer%
   PROCEDURE (charno);
   %creates a partial byte pointer to the character determined by the
   character number passed as argument.  The resulting byte pointer
   needs to have the address of the string (NOT ASTRING) added to
   it.%
   LOCAL wp, cp;
   IF charno = empty THEN RETURN (440700000000B);
   DIV (charno-1) / 5, wp, cp;
   RETURN (chparray[cp] + wp - 1) END.

(apchr) %The calling arguments are a character and the address of an
A-string.  The character gets appended to the end of the A-string.
If the current length of the A-string is the same as the maximum
length (i.e. the string is full) when this routine is called, an
ERROR 6 (exceed capacity) is generated.%

   PROCEDURE (ch, ast);
   REF ast;
   %check for overflow of the string%
   IF (ast.L ← ast.L + 1) > ast.M THEN err($"Exceed capacity");
   al ← chbptr(ast.L) + &ast;
   .al ← a2 ← ch;
   RETURN (ch) END.

(ldchr) %The function of this routine is to load a specified
character from an A-string.  Arguments are the address of the
A-string and the number of the character desired.  The character is
returned.%

   PROCEDURE (ast, cnum);
   IF cnum > [ast].L THEN RETURN(ENDCHR);
   al ← chbptr(cnum) + ast;
   RETURN( .al ) END.

(repchr) %replace a character in an astring%

   PROCEDURE (char, ast, chrno);
   %------------------%
   LOCAL cnt;
   REF ast;
   CASE chrno OF
      > ast.M : err($"Exceed capacity"); %string overflow%
      > ast.L : % blank fill if necessary %
         BEGIN
         IF (cnt ← chrno - ast.L) > 1 THEN apblnk(&ast, cnt-1);
         ast.L ← chrno;
         END;
      ENDCASE;
   al ← chbptr(chrno) + &ast;
   .al ← a2 ← char;
   RETURN (char);
   END.

(apblnk) PROCEDURE(ast, cnt);
```

```
   % Append cnt blanks to the designated astring. %
   REF ast;
   FOR cnt DOWN UNTIL < 1 DO *ast* ← *ast*, SP;
   RETURN END.
```

(apsr) %The purpose of this routine is to append one A-string onto another. Arguments are two A-string addresses. The first string is appended to the second. An ERROR (Exceed Capacity) is generated if the maximum length of the latter string is too short to contain the result.%

```
   PROCEDURE (asfrom, asto);
   LOCAL
      wp,     % word position %
      cp,     % character position %
      bfrom, % byte pointer in source string %
      bto;    % byte pointer in destination string %
   REF asfrom, asto;
   bfrom ← chbmty + &asfrom;
   bto ← chbptr(asto.L) + &asto;
   IF (asto.L ← asto.L + asfrom.L) > asto.M THEN
      err($"Exceed capacity");
   a2 ← asfrom.L;
   UNTIL (a2 ← a2-1) < 0 DO ↑bto ← a1 %use a1% ← ↑bfrom;
   RETURN END.
```

(cpysr) %To copy one A-string into another, call this routine with the address of the source string and the address of thedestination string (in that order). An ERROR is generatedif the maximum length of the destination string is shorterthan the source string.%

```
   PROCEDURE (asfrom, asto);
   REF asfrom, asto;
   CASE asfrom.L OF
      > asto.M : err($"Exceed capacity");
      <= empty : asto.L ← empty;
      ENDCASE
         BEGIN
         asto.L ← asfrom.L;
         mvbfbf(&asfrom+1, &asto+1, (asfrom.L+4)/5);
         END;
   RETURN END.
```

(chpair) %extract substring and append to another string%

```
   PROCEDURE(ast1, lower, upper, ast2);
   LOCAL bto, bfrom, lenstr, lenblank;
   REF ast1, ast2;
   IF upper < lower THEN RETURN;
   lower ← MAX(1, lower);
   CASE ast1.L OF
      < lower :
         BEGIN
         lenstr ← 0;
         lenblank ← upper-lower+1;
         END;
      < upper :
```

```
            BEGIN
            lenstr ← ast1.L-lower+1;
            lenblank ← upper-ast1.L;
            END;
         ENDCASE
            BEGIN
            lenstr ← upper-lower+1;
            lenblank ← 0;
            END;
      IF ast2.L + lenstr + lenblank > ast2.M THEN
         err($"Exceed capacity");
      bto ← chbptr(ast2.L) + &ast2;
      bfrom ← chbptr(lower-1) + &ast1;
      ast2.L ← ast2.L + lenstr + lenblank;
      a2 ← lenstr;
      a3 ← bto;
      a4 ← bfrom;
      UNTIL (a2 ← a2 - 1) < 0 DO ↑a3 ← a1 ← ↑a4;
      a2 ← lenblank;
      a1 ← SP;
      UNTIL (a2 ← a2 - 1) < 0 DO ↑a3 ← a1;
      RETURN END.

(srmake) %make string from value%
      PROCEDURE(value,astrng,base);
      LOCAL power, char;
      REF astrng;
      IF value < 0 THEN
         BEGIN
         IF &astrng THEN *astrng* ← *astrng*, '-
         ELSE apachr('-);
         value ← -value;
         END;
      power ← 1;
      WHILE power*base <= value DO power ← power*base;
      UNTIL power < 1 DO
         BEGIN
         DIV value / power, char, value;
         IF &astrng THEN *astrng* ← *astrng*, char + '0
         ELSE apachr(char + '0);
         power ← power / base;
         END;
      RETURN END.

(srmk) PROCEDURE(value,base);
      %In string construction use apachr to append string to sar.%
      srmake(value, 0, base);
      RETURN END.

(srval) PROCEDURE(astrng, base);
      %convert string to value%
      LOCAL value, cnt, char;
      REF astrng;
      value ← 0;
      cnt ← 1;
      UNTIL (char ← *astrng*[cnt]) = ENDCHR DO
```

```
      BEGIN
      value ← value*base + char - '0;
      BUMP cnt;
      END;
   RETURN (value) END.

(bumpstr)PROC(astr);
   %Increments number in astr by one%
   REF astr;
   %We must use a local string until L10 gets fixed%
      LOCAL STRING tempsr[10];
   *tempsr* ← STRING(VALUE(&astr)+1);
   *astr* ← *tempsr*;
   RETURN;
   END.
(compas) %Compare the contents of two a-strings.  Returns true if the
contents match.%

   PROCEDURE (astr1, astr2);
   LOCAL length, bp1, bp2, count;
   IF (length ← [astr1].L) NOT= [astr2].L THEN RETURN (FALSE);
   bp1 ← bp2 ← chbmty;
   bp1 ← bp1 + astr1;
   bp2 ← bp2 + astr2;
   count ← empty;
   UNTIL (count ← count + 1) > length DO
      IF ↑bp1 # ↑bp2 THEN RETURN (FALSE);
   RETURN (TRUE);
   END.

DECLARE reltab = (0,0,0, %no 0 relational%
   TRUE, FALSE, FALSE, %1: <%
   FALSE, TRUE, FALSE, %2: =%
   TRUE, TRUE, FALSE, %3: <=%
   0, 0, 0, %no 4 relational%
   FALSE, TRUE, TRUE, %5: >=%
   TRUE, FALSE, TRUE, %6: #%
   FALSE, FALSE, TRUE); %7: >%

 (ascom)PROC(astr1, astr2, relation);
   %Accepts addresses of 2 aa-strings, and returns logical vaalue of
   astr1 <relation> astr2%
   LOCAL cmpval, reltbi, minlng, chrcnt;
   REF astr1, astr2;
   reltbi ← relation*3+1;
   minlng ← MIN(astr1.L, astr2.L);
   chrcnt ← 1;
   WHILE minlng >= chrcnt DO IF (cmpval ← *astr1*[chrcnt] -
   *astr2*[chrcnt]) # 0 THEN GOTO nequal ELSE BUMP chrcnt;
   %by here, strings equal...check length%
   cmpval ← astr1.L-astr2.L;
   (nequal): %cmpval < 0 if 1 < 2, 0 if 1 = 2, > 0 if 1 > 2%
   RETURN(reltab[reltbi+sign(cmpval)]);
   END.

 (sign)PROC(value);
```

```
   %returns -1 if vaue < 0, 0 if falue = 0, 1 if value > 0%
   RETURN(IF value < 0 THEN -1 ELSE IF value > 0 THEN 1 ELSE value);
   END.

(repstr) %The routine replaces the characters in one a-string,
OLDAST, with those in another, NEWAST, beginning at CHARNO. If the
new a-string is empty, the routine returns FALSE; otherwise it
returns TRUE.%

   PROCEDURE (newast, oldast, charno);
   %------------------%
   LOCAL astrl, repcnt;
   REF newast, oldast;
   IF (astrl ← newast.L) = empty THEN RETURN (FALSE);
   IF charno + newast.L > oldast.L THEN
      BEGIN
      IF charno + newast.L > oldast.M THEN
         RETURN (FALSE);
      oldast.L ← charno + newast.L;
      END;
   repcnt ← empty + 1;
   UNTIL repcnt > astrl DO
      BEGIN
      *oldast*[charno] ← *newast*[repcnt];
      BUMP charno, repcnt;
      END;
   RETURN (TRUE)
   END.

(srlset)  %To set an A-string to a single character, call this
procedure with the character and the address of the string.%

   PROCEDURE (char, ast);
   [ast].L ← empty;
   apchr (char, ast);
   RETURN END.

(mkbptr) %Make Byte Pointer with the specified position, size and
address.  The address is 23 bits wide to allow indexing and
indirection.%

   PROCEDURE (pos, size, addr);
   al ← addr; !LSHC al,-24;
   al ← size; !LSHC al,-6;
   al ← pos;  !LSHC al,-6;
   RETURN (a2) END.

(slngth) PROCEDURE (bp1, bp2); %string length%
   %returns the lenght of the string  represented by the passed byte
   pointers, the first of which is assumed to be set so that an
   increment and load byte will get first char of the string and the
   second pointing to the last char in the string.  NOTE: the index
   field is ignored and the only the size field of bp1 is used!%
   %bpadr, bpsize, bpbitpos%
   %------------------%
   LOCAL length;
```

```
   length ← (bp2.bpadr - bp1.bpadr)*(36 / bp1.bpsize) +
      (bp1.bpbitpos - bp2.bpbitpos) / bp1.bpsize;
   RETURN(MAX(length, 0));
   END.

DECLARE hshmsk=(
   774B9,
   77776B7,
   7777777B5,
   777777777 4B2,
   777777777776B);

(hash) %An a-string address as argument.  The hash code for that
string is returned.%

   PROCEDURE (ast);
   LOCAL nw, i, gen1, gen2, old1, cp;
   REF ast;
   IF (old1 ← ast.L) <= empty THEN RETURN(0);
   DIV (old1 + 4)/5, nw, cp;
   FOR i ← 1 UP UNTIL = nw DO ast[i] ← ast[i] .A hshmsk[4];
   ast[nw] ← ast[nw] .A hshmsk[cp];
   gen1 ← 0;
   gen2 ← 1;
   UNTIL nw = 0 DO
      BEGIN
      gen1 ← ast[nw] * gen2 + gen1 / 17;
      gen2 ← gen2 * 43;
      nw ← nw - 1;
      END;
   a1 ← gen1;
   !LSH a1,-6;
   RETURN(a1) END.

(bkc) PROCEDURE (astrng);  %backspace char%
   %Given the address of an A-string, this routine will subtract one
   from the length, and then return the last character of the
   string.%
   %-------------%
   REF astrng;
   IF astrng.L <= empty+1 THEN
      BEGIN
      astrng.L ← empty;
      RETURN(ENDCHR)
      END
   ELSE BUMP DOWN astrng.L;
   RETURN(*astrng*[astrng.L]);
   END.

(bkw) PROCEDURE(astr); %backspace word%
   %BackSpace Word (VISIBLE).  The A-string, whose address is passed,
   is backed up one VISIBLE via routine bkc.%
   %-------------%
   LOCAL char;
   CASE char ← bkc(astr) OF
      =SP, =TAB, = CR, = ENDCHR : NULL;
```

```
        ENDCASE REPEAT;
      RETURN(char);
      END.

%Statement scanning & ca support%

   (makeptr)PROC(stid, ptr);
      %Make a pointer pointing to the first character of stid, and store
      it into ptr%
      REF ptr;
      FIND SF(stid) ↑ptr;
      RETURN;
      END.
   (pdc) PROCEDURE (pdcnum, pntloc);
      %pointer decrement. arguments are number of postions to
      move and address of pointer to be decremented.  %

      LOCAL
         end;   %count at end of statement%
      IF scndir = forward THEN
         [pntloc+1] ← MAX([pntloc+1]-pdcnum, 1)
      ELSE
         BEGIN
         cpfse([pntloc]);
         end ← a2;
         [pntloc+1] ← MIN([pntloc+1]+pdcnum, end);
         END;
      RETURN;
      END.

   (dptr) PROCEDURE(pntloc);
      pdc(1, pntloc);
      RETURN END.

   (tstsr) PROCEDURE (ast);
      %Test string.  Compare the T-string specified by SWORK with the
      string whose address is passed as arg.  Fetches characters from
      string by calling READC.  On a successful match does SKIP RETURN
      and SWORK is left pointing to the character after the pattern.  If
      the match fails, normal return.%

      LOCAL
         cnt,    %counter for characters in test string%
         char,   %character from source string%
         tsp;    %pointer into test string%
      REF ast;
      cnt ← ast.L;
      tsp ← chbmty + &ast;
      UNTIL (cnt ← cnt-1) < 0 DO
         IF (char ← READC) = ENDCHR OR ↑tsp # char THEN RETURN;
      SKIP RETURN END.

   (tstf) PROCEDURE (ast);
      %Like tst except looks for any occurence of the test string in the
      remaining portion of the T-string.  This is done by first scanning
      thru looking for the first character of the string, then if find
```

```
   it the rest of the string is?compared.  If get a mismatch SWORK is
   reset to character?after the start of the current partial match
   and the process is repeated.  Failure occurs only when the
   T-string is exhausted.  Returns with SKIP RETURN on success,
   normal return on failure.%

   LOCAL
      cnt,     %counter for characters in test string%
      cpos,    %value of swork1 where got match%
      nchar,   %number of characters in test string%
      char,    %character from source string%
      char1,   %first character of test string%
      tsp,     %pointer into the test string%
      tsp1;    %pointer to first character of test string%
   REF ast;
   nchar ← ast.L;
   IF nchar < 1 THEN SKIP RETURN;
   tsp1 ← chbmty + &ast;
   char1 ← ↑tsp1; %first character of string%
   LOOP %scan until get match for first character in string%
      CASE READC OF
         = ENDCHR : EXIT; %have reached the end%
         = char1 : %have a match on the first character%
            BEGIN
            cpos ← swork1;
            cnt ← 1;
            tsp ← tsp1;
            LOOP
               BEGIN
               IF (cnt ← cnt+1) > nchar THEN SKIP RETURN;
               IF (char ← READC) = ENDCHR THEN EXIT 2;
               IF ↑tsp # char THEN EXIT;
               END;
            %failure -- back to scanning for first character%
            swork1 ← cpos;
            fechc1(scndir, $swork);
            END;
      ENDCASE;
   RETURN;
   END.

(bfs) %branch false and scan %

   % Resets the character position, then reads another character from
   the statement.  If this is not an ENDCHR the character number is
   put on the stack and control is transfered to the location
   specified in the address of the pop.%

   PROCEDURE (brloc);
   swork ← [p-1];
   swork1 ← [p];
   fechc1(scndir, $swork);
   IF READC # ENDCHR THEN
      BEGIN
      [p] ← swork1;
      [m] ← brloc; %change return location%
```

```
      RETURN;
      END;
   p ← p - 2000002B;
   RETURN END.

(fxswork) PROCEDURE;
   fechcl(scndir, $swork);
   RETURN END.

(srsup) PROCEDURE; %dummy procedure for string support code%
   %these are all called by JSP a4,x and return by JRST (a4)%
   (fcp):
      swork ← a1;
      swork1 ← a2;
      !JRST (a4);
   (pcp):
      !PUSH p,swork;
      !PUSH p,swork1;
      !JRST (a4);
   (sptr):
      !MOVE  a2, swork;
      !MOVEM a2,0(a1);
      !MOVE  a2, swork1;
      !MOVEM a2,1(a1);
      !JRST (a4);
   (begarb):
      !PUSH p,swork;
      !PUSH p,swork1;
      !HLRZ a2,a1; %lower bound%
      !PUSH p,a2;
      !HRRZ a2,a1; %upper bound%
      !PUSH p,a2;
      a2 ← 0; %count%
      !PUSH p,a2;
      !JRST (a4);
   (fxsp1): %leave a1 and a2 unchanged%
      sptr1 ← a1;
      a3 ← 1;
      sptr1[a3] ← a2;
      !JRST (a4);
   (fxsp2):
      sptr2 ← a1;
      a3 ← 1;
      sptr2[a3] ← a2;
      !JRST (a4);
   (lpr):
      a1 ← [a2];
      a2 ← [a2+1];
      !JRST (a4);
   END.

EXTERNAL fcp, pcp, sptr, begarb, fxsp1, fxsp2, lpr;

(incarb) PROCEDURE;
   %update position on stack where last succeeded%
   [p-4] ← swork;
```

```
   [p-3] ← swork1;
   %increment count and compare to upper bound%
   RETURN(([p] ← [p]+1) < [p-1]) END.

(endarb) PROCEDURE;
   p ← p - 3000003B;
   scp();
   p ← p - 2000002B;
   RETURN([p+5] IN [[p+3],[p+4]]) END.

(scp) PROCEDURE;
   swork ← [p-1];
   swork1 ← [p];
   fechcl(scndir, $swork);
   RETURN END.

(scn1ft) PROCEDURE;
   fechcl(scndir ← 0, $swork);
   RETURN END.

(scnrht) PROCEDURE;
   fechcl(scndir ← 1, $swork);
   RETURN END.

(chrct) PROCEDURE (char, chrcls); %Character class test%

   CASE chrcls OF
      =1: %CH%
         IF char IN [0,177B] THEN GOTO cctyes;
      =4: %LD%
         IF char IN ['A,'Z] OR
            char IN ['a,'z] OR
            char IN ['0,'9] THEN GOTO cctyes;
      =11: %NLD%
         IF char # ENDCHR AND
            char NOT IN ['A,'Z] AND
            char NOT IN ['a,'z] AND
            char NOT IN ['0,'9] THEN GOTO cctyes;
      =7: %L%
         IF char IN ['a,'z] OR
            char IN ['A,'Z] THEN GOTO cctyes;
      =8: %D%
         IF char IN ['0,'9] THEN GOTO cctyes;
      =9: %PT%
         IF char IN [41B,174B] THEN GOTO cctyes;
      =10: %NP%
         IF char # ENDCHR AND
            char NOT IN [41B,174B] THEN GOTO cctyes;
      =5: %UL%
         IF char IN ['A,'Z] THEN GOTO cctyes;
      =6: %LL%
         IF char IN ['a,'z] THEN GOTO cctyes;
      =2: %ULD%
         IF char IN ['A,'Z] OR
            char IN ['0,'9] THEN GOTO cctyes;
      =3: %LLD%
```

```
            IF char IN [('a,'z] OR
               char IN [('0,'9] THEN GOTO cctyes;
         ENDCASE;
      RETURN (char);
      (cctyes): SKIP RETURN (char)
      END.

   (cct) PROCEDURE(chrcls);
      RETURN(SKIP chrct(READC, chrcls)) END.

   (cctf) PROCEDURE(chrcls);
      WHILE (a1 ← READC) # ENDCHR DO
         IF SKIP chrct(a1, chrcls) THEN RETURN(TRUE);
      RETURN (FALSE) END.

   (span) PROCEDURE(chrcls);
      LOCAL sw1;
      DO sw1 ← swork1 WHILE SKIP chrct(READC, chrcls);
      swork1 ← sw1;
      fxswork();
      RETURN END.

   (tchf) PROCEDURE(char);
      CASE READC OF
         =char: RETURN(TRUE);
         =ENDCHR: RETURN(FALSE);
         ENDCASE REPEAT CASE;
      END.

   (idtst) PROCEDURE(astrng);
      RETURN(getint(swork) = [astrng+1]) END.

   (txtdat) PROCEDURE;
      RETURN(<FILMNP, gettim>(swork)) END.

   (cpfse) PROCEDURE (stid);
      %statement end t-pointer%
      %called by SPL compiler only%

      LOCAL sdbloc, stdb, rng;
      IF stid.stastr THEN
         BEGIN
         a2 ← [stid.stadr].L + 1;
         a1 ← stid;
         RETURN;
         END;
      lodrng(stid : rng);
      stdb ← 0;
      stdb.stfile ← stid.stfile;
      stdb.stpsdb ← [rng].rsdb;
      lodsdb(stdb : sdbloc);
      a2 ← [sdbloc].schars + 1;
      a1 ← stid;
      RETURN END.

%storage manipulation.  stacks, rings, buffers%
```

```
DECLARE FIELD
   systkp = [0(rp), 18:0], %pointer%
   systkt = [0(rp), 13:23], %type (1=stack, 2=ring)%
   systks = [1(rp), 18:18], %size of element%
   systke = [1(rp), 18:0]; %end of store%

(rcpsh) %Record push.  Arguments are (1) address of record, and (2)
address of stack or ring buffer.  The record is pushed on the store.%

   PROCEDURE (recadr, st);
   REF st;
   IF st.systkt NOT IN [1,2] THEN err(0);
   IF (st.systkp ← st.systkp + st.systks) = st.systke THEN
      IF st.systkt = 1 THEN err($"Exceed capacity") %stack overflow%
      ELSE st.systkp ← &st + 2; %ring wrap around%
   IF st.systks = 1 THEN [st.systkp] ← [recadr]
   ELSE mvbfbf (recadr, st.systkp, st.systks);
   RETURN END.

(rpop) %Record pop.  Argument is the address of a store.  Pop the top
element.%

   PROCEDURE (st);
   REF st;
   IF st.systkt = 1 %stack% THEN
      IF st.systkp < &st+2 THEN err(0) %stack underflow%
      ELSE NULL
   ELSE IF st.systkt = 2 %ring% THEN
      IF st.systkp <= &st+2 THEN
         st.systkp ← st.systke
      ELSE NULL
   ELSE err(0); %wrong type%
   st.systkp ← st.systkp - st.systks;
   RETURN END.

(rstsk) %Reset store%

   PROCEDURE (st);
   REF st;
   IF st.systkt NOT IN [1,2] THEN err(0);
   st.systkp ← &st + 2 - st.systks;
   RETURN END.

(rcpto) %Record Pop To.  Arguments are (1) address of store, and (2)
address of record.  Pops the top record on the store to the
designated location.%

   PROCEDURE (st, recadr);
   REF st;
   IF st.systkt NOT IN [1,2] THEN err(0);
   IF st.systks = 1 THEN [recadr] ← [st.systkp]
   ELSE mvbfbf (st.systkp, recadr, st.systks);
   rpop (&st);
   RETURN END.
```

```
(mvbfbf) %This routine moves a buffer of words.  Arguments are
address of source, address of destination, and number of words to
transfer.  It returns the address of the last word into which data
was moved.%

   PROCEDURE (bfrom, bto, nw);
   LOCAL lw;
   IF nw = 0 THEN RETURN;
   lw ← nw + bto - 1; %last word to be transferred to%
   IF bto IN (bfrom, bfrom+nw) THEN
      BEGIN
      a1 ← bfrom;
      a2 ← nw; a3 ← a2;
      a2 ← a2 + a1; %pointer into source%
      a3 ← a3 + bto; %pointer into destination%
      UNTIL (a2 ← a2-1) < a1 DO
         BEGIN
         a3 ← a3 - 1;
         [a3] ← [a2];
         END;
      END
   ELSE %can use block transfer instruction%
      BEGIN
      !HRL a1,bfrom; !HRR a1,bto; !BLT a1,@lw
      END;
   RETURN (lw) END.

%Misc Routines%
   (flagut)PROCEDURE(flagno, jsysno);
      %This routine sets up and calls the flag Jsys', returning the
      appropriate results%
      r1 ← flagno;
      r2 ← [ (CASE flagno OF
         =0: $"JLOCK"; %Journal Lock%
         =1: $"JBFIL"; %Journal Bad File%
         =2: $"SLNKR"; %Auto-start Slinker%
         =3: $"NLSUT"; %auto-start NLS utilty%
         =4: $"OPNLK"; %Open lock lock%
         =5: $"EXPFG"; %Expunge Flag--1 means slinker will expunge%
         =6: $"WMEAS"; %Write out measurements%
         =7: $"IDLOK"; %Identification system lock%
         ENDCASE err($"Illegal Flag Number"))
         +1];
      CASE jsysno OF
         =$tstfg: RETURN (SKIP !JSYS tstfg);
         =$setfg: RETURN (SKIP !JSYS setfg);
         =$rstfg:
            BEGIN
            !JSYS rstfg;
            RETURN(r1);
            END;
         ENDCASE err($"Illegal Flag Jsys");
      END.


FINISH of UTILTY
```

```
<NLS>VERIFY.NLS;1, 1-APR-71 3:02 CHI ;
FILE verify

%control%

   (vfmain) PROCEDURE (fileno, vrfy);
      % values of vrfy %
         % 0 -- Checksum %
         % 1 -- Verify %
      LOCAL
         statsr, %keep each result just for fun%
         statsd,
         statss;
      IF vrfy THEN dismes (1, $"File Verify in progress");
      statsr ← crng(vrfy, fileno);
      IF (statsd ← csdb(vrfy, fileno)) # statsr THEN
      badfil(fileno);
      IF vrfy THEN
         IF (statss ← ckstrc(fileno)) # statsd THEN
         badfil(fileno);
      IF vrfy THEN dismes(0);
      RETURN;
      END.

% procedures for checking data blocks %

   (crng) PROCEDURE (vrfy, fileno);  % ring blocks %
      LOCAL
         stats,   %total number of ring elements in file%
         statb,   %total number of ring elements in block%
         rngblk,  %index in RNGST%
         rn,      %pointer to RNGST entries%
         stid,    %STID for getting blocks loaded by lodrng%
         blkad,   %address of ring block in core%
         nringl,  %number of ring elements in a block%
         frecnt,  %number of elements on the free list%
         freep,   %pointer to free list%
         blkusd;  %used word count for the block%
      REF rn;
      &rn ← filhdr(fileno) + $rngst - $filhed;
      nringl ← (blksiz-fbhdl)/ringl;
      rngblk ← stid ← stats ← 0;
      stid.stfile ← fileno;
      DO BEGIN
         IF rn # 0 THEN
            BEGIN
            stid.stblk ← rngblk;
            lodrng(stid : blkad);
            IF vrfy THEN
               BEGIN
               %check free list%
               frecnt ← 0; %number of entries on the free list%
               IF (freep ← rn.rffree) # 0 THEN
                  BEGIN
                  IF freep NOT IN (fbhdl,blksiz) THEN
                     badfil(fileno);
```

```
                freep ← freep + blkad;
                   %address of start of free list%
                LOOP
                   BEGIN
                   BUMP frecnt; %count entries on free list%
                   %check if have reached end of list%
                   CASE [freep] OF
                      =0: EXIT;
                      NOT IN [fbhdl,blksiz): badfil(fileno);
                      ENDCASE IF frecnt > nringl THEN
                         badfil(fileno);
                      freep ← blkad + [freep];
                      END
                END;
             %check used word count%
             statb ← nringl - frecnt;
             stats ← stats + statb;
             blkusd ← statb * ringl + fbhdl;
             IF blkusd # rn.rfused THEN badfil(fileno);
             END
          END;
       BUMP &rn;
       END
    UNTIL (rngblk ← rngblk+1) = rngm;
    RETURN (stats) END.

(csdb) PROCEDURE(vrfy, fileno); % statement data blocks %
    LOCAL
       stats,  %number of nongarbage SDB's in file%
       sdbblk, %index in DTBST%
       dt,     %pointer to DTBST entry%
       stdb,   %STDB for getting blocks loaded by lodsdb%
       blkad,  %address of the block%
       freep,  %address of free space%
       sdbpt,  %pointer to SDB's in the block%
       sdbusd; %used word count in the block%
    REF dt;
    &dt ← filhdr(fileno) + $dtbst - $filhed;
    sdbblk ← stdb ← stats ← 0;
    stdb.stfile ← fileno;
    DO BEGIN
       IF dt # 0 THEN
          BEGIN
          stdb.stblk ← sdbblk;
          lodsdb(stdb : blkad);
          IF vrfy THEN
             BEGIN
             %check free space pointer%
             IF (freep ← dt.rffree)
                NOT IN [fbhdl,blksiz] THEN badfil(fileno);
             freep ← freep+blkad; %address of free space%
             sdbpt ← blkad+fbhdl; %address of first SDB%
             sdbusd ← fbhdl; %used word count%
             WHILE freep > sdbpt DO
                BEGIN
                IF NOT [sdbpt].sgarb THEN
```

```
                    BEGIN
                    sdbusd ← sdbusd + [sdbpt].slength;
                    BUMP stats;
                    END;
                 %address of next SDB%
                 IF (sdbpt := sdbpt + [sdbpt].slength) >= sdbpt THEN
                 badfil(fileno);
                 END;
              IF sdbusd # dt.rfused THEN badfil(fileno);
              END;
           END;
        BUMP &dt;
        END
     UNTIL (sdbblk ← sdbblk+1) = dtbm;
     RETURN (stats) END.

% procedure for checking structure %

  (ckstrc) PROCEDURE(fileno);
     LOCAL
        org,    %STID for origin of file%
        stats,  %number of statements in structure%
        cur,    %STID of current statement%
        cursup, %STID of up of current statement%
        clhead, %head flag value expected for cur%
        nxtcur, %STID of next statement%
        sdb,    %location of SDB%
        curlev; %level in structure of cur%
     REF sdb;
     cur ← curlev ← stats ← 0;
     cur.stfile ← fileno;
     cur.stpsid ← origin;
     org ← cursup ← cur;
     RESET fvstk;
     PUSH org ON fvstk;
     clhead ← TRUE;
     LOOP %once thru for each statement%
        BEGIN
        BUMP stats;
        %check the sdb%
        fchsdb(cur : &sdb);
        IF sdb.sgarb OR
           sdb.spsid # cur.stpsid THEN badfil(fileno);
        %check head flag%
        IF clhead # getfhd(cur) THEN badfil(fileno);
        IF (nxtcur ← getsub(cur)) # cur THEN
           BEGIN %go down in structure%
           IF (curlev ← curlev+1) > 30 THEN badfil(fileno);
           PUSH cursup ON fvstk;
           cursup ← cur;
           cur ← nxtcur;
           clhead ← TRUE;
           END
        ELSE %go to successor%
           BEGIN
           nxtcur ← getsuc(cur);
```

```
            WHILE nxtcur = cursup AND nxtcur # org DO
               BEGIN
               %must have tail flag set%
               IF getftl(cur) = FALSE THEN badfil(fileno);
               cur ← nxtcur;
               nxtcur ← getsuc(cur);
               POP fvstk TO cursup;
               curlev ← curlev-1;
               END;
            cur ← nxtcur;
            %head flag must be off%
            clhead ← FALSE;
            END;
         IF cur = org THEN RETURN (stats);
         END;
      END.

FINISH of verify
°°p] OF
   =0: EXIT;
   NOT IN [fbhdl,blksiz): badfil(fileno);
   ENDCASE IF frecnt > nringl THEN
      badfil(fileno);
freep ← blkad + [freep];
END
END;
%check used word count%
statb ← nringl - frecnt;
stats ← stats + statb;
blkusd ← statb * ringl + fbhdl;
IF blkusd # rn.rfused THEN badfil(fileno);
END
END;
BUMP &rn;
END
UNTIL (rngblk ← rngblk+1) = rngm;
RETURN (stats) END.

(csdb) PROCEDURE(vrfy, fileno); % statement data blocks %
```