UNECLASSIFIED NPS92-80-012

Baseline implementations of the Standard Line Editor (SLED)
Baseline Implementations of the Standard Line Editor (SLED)

L. Cox, R. Coulter, C. Taylor, R. Burnham, and S. Smart

October 1980

Approved for public release; distribution unlimited.

Prepared for Naval Postgraduate School, Monterey California 93940
DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.
In response to a recognized requirement for a more uniform man machine interface, especially in multiple machine networks, a standardized text editor was proposed (1). This editor, SLED, was designed to be easily implementable in several commonly available higher level languages. This document reviews two baseline implementations taken directly from the SLED standards which users may want to consider when implementing SLED upon local systems. These baseline programs were written and documented with portability and understandability as goals.
Baseline Implementations of the
Standard Line Editor (SLED)

Lyle A. Cox Jr.
Editor

Department of Computer Science
Naval Postgraduate School
Monterey, California

ABSTRACT

In response to a recognized requirement for a more uniform man machine interface, especially in multiple machine networks, a standardized text editor was proposed (1). This editor, “SLED” was designed to be easily implementable in several commonly available higher level languages. This document reviews two baseline implementations taken directly from the SLED standards which users may want to consider when implementing SLED upon local systems. These baseline programs were written and documented with portability and understandability as goals.

BACKGROUND

During the Winter of 1979 and Spring of 1980, shortly after the Standard Line Editor definition was first developed, several persons at the Naval Postgraduate School undertook implementations of SLED. Two of these implementations -- the documentation and the code -- are reproduced here.

Appendix A contains the SLED implementation developed by C. F. Taylor, Jr. on an IBM 360 system in FORTRAN. Appendix B contains an alternate implementation developed by R. M. Burnham, R. J. Coulter, and S. W. Smart in the PASCAL language.

Neither of these implementations should be considered "off the shelf" commercial quality software ready for installation. These systems do however provide two critical portions of the implementation:

1. Each contains the basic code in a "portable" higher level language, and

2. The code of each implementation is written for readability.
and the documentation has been written to allow the program to be adapted to any system with a minimum of difficulty.

It is hoped that these baseline programs will serve to facilitate the implementation of the SLED man-machine interface on a variety of machines.

Acknowledgments:

I would like to thank the authors of the included codes, C. Taylor, R. Burnham, R. Coulter, and S. Smart for their particularly outstanding work on this project. I would also like to acknowledge the assistance of many of their contemporaries who attempted alternative implementations, and who helped in the evolution of the SLED standards. I would also like to thank Dr. R. W. Hamming and LTCOL. R. R. Schell for their comments and interest.

Lyle A. Cox Jr.
15 August, 1980

Appendix A
SLED FORTRAN Implementation
(by C. F. Taylor Jr.)

The purpose of these notes is to briefly describe the accompanying FORTRAN implementation of SLED.

SLED FORTRAN Version FORT1.1 was implemented on an IBM 360/67 computer under the CP/CMS time-sharing operating system at the W. R. Church Computer Center at the Naval Postgraduate School, in a superset of IBM FORTRAN IV, Level G, which included the "IF-THEN-ELSE" and "WHILE-DO" constructs of WATFIV-S. The addition of these two constructs greatly simplified the writing of the program and were implemented with the aid of a preprocessor written by this author. Standard FORTRAN version (the output of the preprocessor) is shown on the following pages.

This version was compiled using an IBM FORTRAN IV/G compiler.

The package was implemented using the utility routines shown on pages A37-A40 of the listings. The 'TDISK EXEC' module was used to obtain the required disk space for the 4000 line (320K byte) temporary work file. It is assumed that this work space would be provided at the system level in any actual implementation of SLED. Because it is a direct-access disk file, sufficient space must be available in advance for the maximum capacity of the editor, which in this case is 4000 lines. This figure was selected somewhat arbitrarily; CP/CMS gives the user 800 bytes per track (IBM 2314), using the remaining space for overhead, so at 80 bytes per line, 4000 lines represents 400 tracks. Installations which do not need to edit such large files could reduce disk requirements by further limiting the capacity of the editor as follows:

1. Do a global substitution to replace the string '4000' in the program by the new capacity.
2. Modify the 'DEFINE FILE' statement in subroutine MEMORY as necessary.
3. Alter the FILEDEF statement in 'SLED1 EXEC' (or the equivalent action on another system) to request less disk space.

The basic data structures used are as follows: The file to be edited (unless a new file is being established) is read in sequentially by subroutine OPEN and stored in the work file (described above) which is conceptually a 4000 line by 80 column array. All references to the work file are made through calls to subroutine MEMORY. The lines of the work file are not necessarily kept in order. A 4000 element from the file is determined by popping the top element from the stack. This value is then recorded in the 4000 element array LPTR. LPTR(I) then always gives the address in the work file of the Ith line of the text file. As lines are deleted, their addresses are pushed back onto
additions and deletions of lines by manipulating pointers rather than the text itself. Still, the work file may be accessed sequentially by using LPTR(I) as the index.

Input from the terminal is buffered in a circular queue in subroutine GETLIN. This permits the "stacking" of more than one command per line at the terminal. Calls to GETLIN return a line from the head of the queue or, if the queue is empty it reads in a new line from the terminal.

The QUIT subroutine writes the work file sequentially (using LPTR as an index) to the output file.

Additional notes which may be of interest to the local implementor follow: (including deviations from the SLED standard).

1. Integer*2 variables were used for all character storage (and for the two large arrays STACK and LPTR). Only one character was stored per word.

2. Because FORTRAN reads only fixed-length formatted records from the terminal, the carriage return cannot be used to terminate a string. This means that 'DS' and 'RS' commands must use the logical terminator character (default '$') to terminate strings and that the 'RS' command should be used as follows: RS$str1$str2$.

3. The 'RS' command replaces only the first occurrence of a string in a line because of this author's firm conviction that to allow only multiple substitution would be dangerous. In an editor such as this without a TAB function, a common string substitution would be to replace one blank with two blanks on a line. What would happen if this were done for every occurrence of a blank on the line is too horrible to contemplate.

4. The CP/CMS operating system and IBM FORTRAN required that filenames be handled external to the program itself. The routine 'SLED EXEC' executes the simple program 'SLEDVERS' when 'SLED' is typed to alert the user to the required entry procedure, 'SLEDI <filename> <file-type>'. SLEDI EXEC then invokes the actual edit program, SLED2. The filename given the program internally is meaningless and provided only for cosmetic reasons. This system requires that only one file be opened per session.

5. Another limitation of FORTRAN required that the program read input from the line following the program prompt, not a serious problem.
SLED - STRUCTURED FORTRAN
FILE: SLED FORTRAN III

NAVAL POSTGRADUATE SCHOOL

STANDARD LINE EDITOR--FORTRAN IMPLEMENTATION

SLED VERSION FORTL. NPS MONTEREY 900401
PROGRAMMED BY: C. F. TAYLOR, JR., CODE 55TA

FOR FURTHER INFORMATION SEE NPS TECHNICAL REPORT
NPS 52-80-001 BY L. A. COX, JR.

MAIN PROGRAM:

READS IN COMMANDS AND CALLS THE APPROPRIATE SUBROUTINE
IMPLEMENTS THE "EDIT" MODE

COMMON /BLK1/IN,OUT
1 INTEGER*2 CI,C2,AG,DL,M,Q,R,S,T,V INLINE,BLNK
    INTEGER IN,OUT,ERRCT,CURLIN,NC
    DIMENSION INLINE(180)
    LOGICAL FLAG,OPENFL,MFLAG
    DATA A/*A*/,'C'/C/*D*/,'L'/*L*/,'M'/*M*/,'Q'/*Q*/,'R'/*R*/,'T'/*T*/,'V'/*V*/,'BLNK'/T */
2000 FORMAT (' '2X')
2010 FORMAT ('-INVALID COMMAND- ',2A1)
2020 FORMAT ('-NO TEXT FILE OPEN-')
2030 FORMAT ('-ONLY ONE FILE CAN BE OPENED PER SESSION IN THIS ',
1 'VERSION OF SLED-')
C EDIT MODE--WRITE PROMPT
    WRITE (OUT,2000)
    CALL GETLIN(INLINE,NC)
    CI = INLINE(1)
    C2 = INLINE(2)
C FLAG GOES FALSE AFTER 'QUIT'
C FLAG = .TRUE.
C OPENFL GOES TRUE AFTER FILE IS OPENED
    OPENFL = .FALSE.
C MAIN EDIT LOOP
    WHILE (FLAG) DO
1 IF (.NOT.(OPENFL .OR. (CL.EQ.0) .OR. (CL.EQ.M))
     *OR (C1.EQ.V)) THEN DO;
     FILE NOT OPEN AND THIS IS NOT AN 'OPEN' COMMAND
     OR A 'MENU' OR 'VERSION' REQUEST
     WRITE (OUT,2020)
     ELSE DO
     EMULATE CASE (SWITCH) STATEMENT TO PROCESS COMMANDS
     IF (CI = EQ. BLNK) GO TO 290
     IF (CI.NE.L) GO TO 10
     CALL LIST(INLINE)
     CALL SCREEN(INLINE)
     GO TO 200
10 CONTINUE
     IF (CI.NE.S) GO TO 20
     GO TO 200
20 CONTINUE
     IF ((CI.NE.R).OR.(C2.NE.S)) GO TO 30
CALL RS(INLINE)
GO TO 200

30 CONTINUE
IF (C1.NE.R) OR (C2.NE.L) GO TO 40
CALL RL(INLINE)
GO TO 200

40 CONTINUE
IF (C1.NE.A) OR (C2.NE.L) GO TO 50
CALL AL(INLINE)
GO TO 200

50 CONTINUE
IF (C1.NE.D) OR (C2.NE.S) GO TO 60
CALL US(INLINE)
GO TO 200

60 CONTINUE
IF (C1.NE.Q) GO TO 70
CALL QUIT
FLAG = .FALSE.
GO TO 200

70 CONTINUE
IF (C1.NE.V) GO TO 80
CALL VERS
GO TO 200

80 CONTINUE
IF (C1.NE.M) GO TO 90
CALL MENU
GO TO 200

90 CONTINUE
IF (C1.NE.O) GO TO 100
IF (OPENFL) THEN DO
END IF
GO TO 200

100 CONTINUE
IF (C1.NE.C) OR (C2.NE.T) GO TO 110
CALL CT(INLINE)
GO TO 200

110 CONTINUE
C
IF PROGRAM GETS HERE, COMMAND IS INVALID
WRITE (OUT,2030) CL,C2
ERRCT = ERRCT + 1
C
200 END CASE
CONTINUE
IF (ERRCT.GE.2) THEN DO
CALL MENU
ERRCT = 0
END IF
END IF

C
IF (FLAG) THEN DO
GET NEXT LINE
IF (.NOT.MFLAG) WRITE (OUT,2000)

FILE: SLED FORTRAN TL

CALL GETLINE(INLINE, NC)
C1 = INLINE(1)
C2 = INLINE(2)
END IF
END WHILE
STOP
END

SUBROUTINE LIST(CLINE)

DISPLAYS TEXT TO THE TERMINAL

COMMON /BLK1/ IN, OUT
COMMON /BLK2/ LPTR, MAXLIN, EOF
COMMON /BLK3/ TCHAIN, ERRCT, CURLIN
COMMON /BLK4/ BLANK, LPTR, COMMA, OUTLIN, CLINE
COMMON /BLK5/ MFLAG, ERRCT, CURLIN

INTEGER IN, OUT, I, J, N1, N2, MAXLIN, EOF, ERRCT, CURLIN, FETCH
LOGICAL MFLAG, EFLAG

DIMENSION CLINE(80), OUTLIN(80), LPTR(400)

FORMAT (* -INVALID COMMAND-*)
FORMAT (* *1X, 1X, 80A1)
FORMAT (* *-EOF-*)

IF COL 2 IS BLANK, PRINT CURLIN AND EXIT
IF (CLINE(2) = BLANK) THEN DO
    CALL MEMORY(FETCH, OUTLIN, LPTR(CURLIN))
    WRITE (OUT, 2110) CURLIN, OUTLIN
ELSE DO
    CALL COMLIN2, CLINE, N1, N2, EFLAG
    IF (N1 = 0) N1 = 1
    IF (N1 GT N2) EFLAG = .TRUE.
    IF (N1 GE EOF) EFLAG = .TRUE.
    IF (EFLAG) THEN DO
        ERRCT = ERRCT + 1
        WRITE (OUT, 2100)
    ELSE DO
        ERRCT = 0
        I = N1

FILE: SLED FORTRAN TI  NAVAL POSTGRADUATE SCH'01

WHILE (1.LE.N2).AND. (I.LT.EOF)) DO
  CALL MEMORY (FETCH, OUTLIN, LPTR(1))
  WRITE (OUT,2150) I, OUTLIN
  CURLIN = I
  I = I + 1
END WHILE
IF (1.GE.EOF) WRITE (OUT,2120)
END IF
RETURN
END

SUBROUTINE SCREENICLINE)

C Displays 20 lines beginning with CURLIN or other specified line

COMMON /BLK/, IN, OUT
/BLK2/ LPTR, MAXLIN, EOF
/BLK5/ MPLAG, ERRCT, CURLIN
INTEGER IN, OUT, MAXLIN, EOF, ERRCT, CURLIN, I, N1, N2, LIMIT,
INTEGER CLINE, LPTR, OUTLIN, BLNK
LOGICAL MFLAG, EFLAG
DIMENSION CLINE(80), OUTLIN(80), LPTR(400)
DATA BLNK/* ' ', FETCH/O/
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
FORMAT (* -INVALID COMMAND-')
FORM (X,80A1)
FORMAT (* -EOF-')
C
EFLAG = .FALSE.
C  Find OUT whether user specified a line
IF (CLINE(2), NE, BLNK) THEN DO
  CALL COMLIN(2, CLINE, N1, N2, EFLAG)
  IF (N1 .LE. 0) N1 = 1
  IF (N1 .GE. EOF) EFLAG = .TRUE.
  IF (*MUL.EFLAG) CURLIN = N1
END IF
IF (EFLAG) THEN DO
  ERRCT = ERRCT + 1
  WRITE (OUT,2100)
ELSE
  ERRCT = 0
  LIMIT = MIN (CURLIN + 19, EOF - 1)
  DO 10 I = CURLIN, LIMIT
    CALL MEMORY (FETCH, OUTLIN, LPTR(1))
  10 CONTINUE
  CURLIN = LIMIT
IF (LIMIT .EQ. EOF - 1) WRITE (OUT,2120)
END IF
RETURN
END
SUBROUTINE ALICLINE

REPLACES CURRENT LINE OR THE SPECIFIED LINE OR LINES WITH
ANY NUMBER OF LINES

COMMON /BLK1/ IN,OUT
/BLK2/ LPTR,MAXLN,EOF
/BLK3/ FFLAG,ERRCT,CURLIN

INTEGER IN,OUT,MAXLN,EOF,ERACT,CURLIN,N1,N2,I,J,LIMIT,
STORE

INTEGER*2 CLINE,LPTR,BLNK
LOGICAL *FLAG,EFLAG
DIMENSION CLINE(8000),LPTR(4000)
DATA BLNK/11,11/STORE/11/

FORMAT (11-'INVALID COMMAND-
N1 = CURLIN
N2 = N1

Determine which line(s) to replace
IF (CLINE(1) .NE. BLNK) THEN DO
CALL COMLIN(N1,CLINE,N1,N2,EFLAG)
IF (N1 .LE. 0) EFLAG = .TRUE.
IF (N1 .GE. EOF) EFLAG = .TRUE.
IF (.NOT. EFLAG) CURLIN = 'N1
END IF
IF (EFLAG) THEN DO
ERRCT = ERRCT + 1
WRITE (OUT,2100)
ELSE DO
ERRCT = 0
IF (N2 .GE. EOF) N2 = EOF - 1
REMOVE DESIGNATED LINES
N = N2 - N1 + 1
DO 20 I = 1,N
LIMIT = EOF - 2
CALL PUSHLTR(N1)
DO 10 J = N1,LIMIT
L1PTR(J) = LTR(J+1)
20 CONTINUE
EDF = EOF - 1
CONTINUE
NOW INPUT REPLACEMENT LINES
CALL INPUT
END IF
RETURN

END

SUBROUTINE ALICLINE

INPT TEXT AFTER LINE V

COMMON /BLK1/ IN,OUT

11
FILE: SLED FORTRAN T1 NAVAL POSTGRADUATE SCHOOL

 INTEGER*2 BLK,C LINE
 INTEGER IN,OUT,J,L,N,N1,N2,ERROR,CURLIN
 LOGICAL MFLAG,EFLAG
 DIMENSION CLINE(80)
 DATA BLNK/1/

 FORMAT ('-INVALID COMMAND-')

 C
 C EXTRACT LINE NUMBER FROM COMMAND LINE
 CALL COMLIN3,CLINE,N1,N2,EFLAG)
 IF (N1 .LT. 0) EFLAG = .TRUE.
 N = N1
 IF (EFLAG) THEN
 E#:CT = ERRCT + 1
 WRITE (OUT,2100)
 ELSE DO
 ERRCT = 0
 CURLIN = "N + 1"
 CALL INPUT
 END IF
 RETURN
 END

 SUBROUTINE OS(CLINE)

 PART OF SLED PACKAGE
 DISPLAYS ALL LINES CONTAINING THE DESIGNATED STRING, POSSIBLY
 LIMITED TO LINES N THROUGH M.

 COMMON /BLKI/ IN,OUT
 /BLK2/ LPTR,M AXLIN,E0F/
 /BLK3/ MFLAG,ERROR,CURLIN
 INTEGER IN,OUT,M AXLIN,E0F,ERROR,CURLIN,N1,N2,NC,FETCH,MC1
 INTEGER*2 CLINE,LPTR, BLNK,STRING,LINE
 LOGICAL MFLAG,EFLAG,FOUND
 DIMENSION CLINE(80),LINE(80),LPTR(4000),STRING(80)
 DATA BLNK/1/,FETCh/0/

 FORMAT ('-INVALID COMMAND-')

 FORMAT (14*I,1X,80A1)
 FORMAT (14*I,1X,80A1)
 FORMAT ('-NO STRING FOUND-')

 C
 C DEFAULT VALUES
 FOUND = .FALSE.
 EFLAG = .FALSE.
 N1 = 1
 N2 = EOF = 1

 C DETERMINE WHETHER N1,N2 WERE SPECIFIED BY USER
 IF (CLINE(3) .NE. BLNK) THEN DO
 CALL COMLIN3, CLINE,N1,N2,EFLAG)
 CALL IF (N1 .LE. EOF) EFLAG = .TRUE.
 IF (N2 .GE. EOF) N2 = EOF - 1
 IF (N1 .LE. 0) N1 = 1
 END IF
FILE: SLED FORTRAN T1

IF (EFLAG) THEN DO
   ERRC = ERRC + 1
   WRITE (OUT,2100)
ELSE DO
   ERRC = 0
   FETCH STRING; ISSUE PROMPT IF NECESSARY
   IF (.NOT. MFLAG) WRITE (OUT,2220)
   CALL GETLIN (STRING,NC)
   IF (NC .LE. 0) THEN DO
      ERRC = ERRC + 1
      WRITE (OUT,2100)
   ELSE DO
      DO 20 1 = NL,N2
         CALL MEMORY(FETCH,LINE,LPTR(I))
         CALL SEARCH(LINE,STRING,NC,MATCH,MC)
         IF (MATCH) THEN DO
            FOUND = *TRUE*
            WRITE (OUT,2110) I,LINE
            CURLIN = I
         END IF
      CONTINUE
      IF (.NOT. FOUND) WRITE (OUT,2250)
   END IF
   RETURN
END IF
END SUBROUTINE RS(CLINE)

PART OF SLED PACKAGE
REPLACES THE FIRST OCCURRENCE OF STRING1 WITH STRING2 ON THE
CURRENT LINE OR WITHIN THE SPECIFIED RANGE OF LINES

COMMON /BLK1/ IN,OUT
/BLK2/ LPTR,MAXLIN,EOF*
/BLK3/ MFLAG,ERRCT,CURLIN
INTEGER IN,OUT,MAXLIN,EOF,ERRCT,CURLIN,NL,N2,I,J,K,L,Y,
N,MC1,MC2,FETCH,STORE
INTEGER CLINE,LPTR,BLNK,STR1,STR2,LINE
DIMENSION LPTR(4000),CLINE(80),STR1(90),STR2(90),LINE(90)
DATA BLNK/* */ ,FETCH/O/,STORE/* /
LOGICAL MFLAG,FOUND,MATCH,EFLAG

2100 FORMAT (* -INVALID COMMAND*-)
2110 FORMAT (* -OLD STRING?*
2230 FORMAT (* -NEW STRING?*
2240 FORMAT (* -NO STRING FOUND*-)

DEFAULT CONDITIONS
NI = CURLIN
N2 = NI
EFLAG = *.FALSE.*

INTERPRET COMMAND LINE
IF (CLINE(1),NE.,BLNK) THEN DO
   CALL COM(LINE,NI,N2,EFLAG)
   IF (NI .LE. 0) EFLAG = *.TRUE.*
FILE: SLED FORTRAN T1

IF (NL .GE. EOF) EFLAG = .TRUE.
IF (N2 .GE. EOF) N2 = EOF - 1
END IF
IF (EFLAG) THEN DO
ERRCT = ERRCT + 1
WRITE (OUT, 2100)
ELSE DO
ERRCT = 0
WRITE (OUT, 2100)
END IF
READ IN TWO STRINGS; PROMPT IF NECESSARY
IF (.NOT. *EFLAG) WRITE (OUT, 2230)
CALL GETLIN(STRL,NCL)
IF (NC1 .LE. 0) EFLAG = .TRUE.
IF (.NOT. *EFLAG) WRITE (OUT, 2240)
IF (EFLAG) THEN DO
ERRCT = ERRCT + 1
WRITE (OUT, 2100)
RETURN
END IF
NOW FIND STRING1
FOUND = .FALSE.
DO 90 K = N1,N2
CALL MEMORY(FETCH,LINE,LPTR(K))
CALL SEARCH(LINE,STR1,NCL,MATCH,NC1)
IF (MATCH) THEN DO
NOW MAKE SUBSTITUTION
J = NC1
FOUND = .TRUE.
DELETE STRING1
DO 20 I = 1,NC1
DO 10 M = J,J+79
LINE(M) = LINE(M+1)
10 CONTINUE
20 CONTINUE
NOW MAKE ROOM FOR STRING2
IF (NC2 .GT. 0) THEN DO
DO 40 L = 1,NC2
M = 81 - J
DO 30 I = 2,9
LINE(81-I) = LINE(82-I)
30 CONTINUE
40 CONTINUE
NOW INSERT NEW STRING
DO 45 I = 1,NC2
LINE(J+I-1) = STR2(I)
45 CONTINUE
END IF
STORE REVISED LINE
CALL MEMORY(STORE,LINE,LPTR(K))
DISPLAY REVISED LINE
WRITE (OUT, 2100) K,LINE
REMOVE NEW IN CC 1 OF ABOVE LINE TO ENABLE
DISPLAY OF EACH LINE IN WHICH A STRING HAS BEEN
REPLACED
CURLIN = K
FILE: SLED  FLTRAN TI

NAVAL POSTGRADUATE SCHOOL

SUBROUTINE CT

PART OF SLED PACKAGE
CHANGES THE MESSAGE TERMINATOR TO ANY VALID CHARACTER

COMMON / BLK1/ INT,OUT
          /BLK4/ TCHR
          /BLK5/ *FLAG,ERCT,CURLIN
           INTEGER I,OUT,ERCT,CURLIN,NC
           INTEGER*2 TCHR,INLIN,RLNK
           LOGICAL *FLAG
           DIMENSION INLIN(80)
           DATA BLK1(100)
            FORMAT (13JX,*INVALID COMMAND*!)
            FORMAT (* TERMINATOR?*)

IF (.NOT. *FLAG) THEN DO
   ISSUE PROMPT
   WRITE (OUT,2200)
   END IF
   CALL GETLIN(INLIN,NC)
   IF (((NC.EQ.01.OR.(INLIN(1).EQ.BLNK)) THEN DO
      ERCT = ERCT + 1
      WRITE (OUT,2100)
   ELSE DO
      TCHR = INLIN(1)
   END IF
   RETURN
END SUBROUTINE MENU

PART OF SLED PACKAGE
PROVIDES USER WITH A SUMMARY OF AVAILABLE COMMANDS
AND THEIR FORMATS

COMMON / BLK1/ INT,OUT
               INTEGER IN,OUT
            200 FORMAT (13JX,SLED COMMAND SUMMARY:*//13LINE/TEXT INSERT",T39,
                *string replacement",T3X,ALT"T10,*INSERT AFTER LINE","T3X,
                *replace <STRING">/T3X,"<PLACE STRING"/T3X,"<REPLACE STRING AFTER LINE","T3X,"OR"
                *string search","T3X,"<<STRING">/T3X,"<DISPLAY CURRENT STRING">/T3X,"<DISPLAY LINES "/T3X,"LINES","T40,"THRU","T40,"OR LINE","T40,"SHOW ANY LINES")
                *with <STRING">/T3X,"LINES","T40,"THRU","T40,"OR SHOW ANY LINES")
            201 FORMAT (13JX,*SCREEN* OF LINES","T50,"N-M CONTAINING *P*/T3X,"SN",T10),
               *X*
FILE: SLED FORTRAN TI

SUBROUTINE VERS

C

C PAP'T OF SLEO PACKAGE

COMMON /BLK1/ IN,OUT
INTEGER IN,OUT
220 FORMAT (* SLED VERSION FORTRAN NPS MONTEREY 800401)*
1 LOCAL EXPERT IS C. TAYLOR 408-664-2891 0900-1700 PST/PUT*
2 * LINE DELETE KEY IS < > (ASCII) OR <CENT SIGN> '*
3 * (EBCDIC)*' CHARACTER DELETE KEY IS <#> '*
4 * EDITOK LOGICAL MESSAGE TERMINATORS ARE:*
5 * (1) <RETURN> AND (2) <$>
6 * AND CAN BE CHANGED TO ANY STANDARD FORTRAN CHARACTER. '*
7 * ALL INPUT IS TRANSLATED TO UPPER CASE. '*
8 * THE FOLLOWING DEVIATIONS FROM SLED STANDARD WERE REQUIRED: ')
221 FORMAT (*
9 * (1) THE UNIVERSAL ENTRY COMMAND "SLED" INVOKES INSTRUCTIONS*
10 * FOR A NON-STANDARD ENTRY: "SLED" <FILENAME> <FILETYPE>
11 * ONLY THE FILE PERSN SESSION CAN BE OPENED. '*
12 * (3) MAXIMUM FILE SIZE IS 4000 LINES. '*
13 * (4) THE USER IS ASKED TO INDICATE WHETHER HE IS EDITING A *
14 * NEW FILE IN ORDER TO PREVENT A DISK READ ERROR IN FORTRAN*
15 * WHEN <RETURN> IS USED AS A LOGICAL MESSAGE /
16 * TERMINATOR, THE LINE OR STRING IS PADDED WITH 'BLANKS UN'*
17 * THE RIGHT. THIS AFFECTS THE RS FUNCTION ONLY. '*
222 FORMAT (*
18 * (6) ONLY THE FIRST OCCURRENCE OF A STRING IN EACH LINE*
19 * IS REPLACED TO PERMIT FREE SUBSTITUTIONS OF 'BLANKS*'
20 * WRITE (OUT,220)
21 * WRITE (OUT,221)
22 * WRITE (OUT,222)
RETURN
END SUBROUTINE OPEN

COMMON /BLK1/ IN,OUT

OPEN TEXT FILE AND WORKSPACE FILE

READS TEXT FILE INTO WORKSPACE IF IT ALREADY EXISTS

INITIALIZES POINTERS ETC.

COMMON /BLK1/ IN,OUT

/BLK3/ LTRN,MAXLIN,EOF
2/BLK6/ LTRN,MAXLIN,EOF
/BLK5/ FILELINE,ERRCT,CURLIN
/BLK6/ STACK,STKPTR

INTEGER IN,OUT,TFIL,LINE,ERRCT,CURLIN,STORE,MAXLIN,EOF,
FILE: SLED   FORTRAN II

1

INTEGER*4 LPT, STACK, FNAME, INLIN, YES, NO, REPLY, LINE
LOGICAL *4 FREE

DIMENSION LPT(4000), BLINE(80), STACK(4000), FNAME(80), INLIN(80)
DATA YES, 'Y', BLINE/80* 'Y' , 'N', 'N*', 'STOR' /1/

100 FORMAT (90A1)
2040 FORMAT (I5, ' LINES IN FILE: ', 90A1)
2360 FORMAT (I5, 'CREATING FILE: ', 90A1)
2400 FORMAT (I5, 'FILENAME?')
2420 FORMAT (I5, 'IS THIS A NEW FILE?')

C

C INITIALIZE
MAXLIN = 4000
CURLIN = 1

C READ IN FILENAME (COSMETIC)
IF ( .NOT. 4FLAG) WRITE (OUT,2400)
C CALL GETLINIFNAME,NC
C ASK WHETHER IT IS A NEW FILE (TO PREVENT FORTRAN READ ERROR)
WRITE (OUT,2410)
READ (IN,1000) INLIN
REPLY = [INLIN(1)]
IF (REPLY .NE. YES) AND. (REPLY .NE. NO) GO TO 5
IF (REPLY .EQ. YES) THEN DO
WRITE (OUT,2050) FNAME
EOF = 1
STKPTR = 1
CALL ACTIVATE FILE WITH AN ACCESS
LPTR(1) = 1
CALL MEMORY (STORE, BLINE, LPTR(1))
ELSE DO:
NOW READ IN TEXT FILE
LINE = 0
WHILE (.TRUE.) DO
READ (TFIL) INLIN
LPTR(LINE+1) = LINE + 1
CALL MEMORY (STORE, INLIN, LPTR(LINE+1))
LINE = LINE + 1
END WHILE
CONTINUE
IF (LINE .GE. MAXLIN) THEN DO
WRITE (OUT,2420)
STOP
END IF
STKPTR = LINE + 1
EOF = STKPTR
C TELL USER FILE OPEN
WRITE (OUT,2040) LINE, FNAME
END IF
DO 20 I = 1, MAXLIN
STK(1) = I
20 CONTINUE
RETURN
END
SUBROUTINE QUIT

17
FILE: SLED FORTRAN TI

NAVAL POSTGRADUATE SCHOOL

PART OF THE SLED PACKAGE
CLOSES OUT THE WORK FILE AND WRITES THE NEW OR UPDATED
TEXT FILE

COMMON /BLK1/ LPTR,MAXLIN,EOF
COMMON /BLK3/ TFLE
/BLK1/ IN,OUT
INTEGER MAXLIN,IN,OUT,EOF,TFLE,L,LIMIT
INTEGER*2 LINE,LPTR
DIMENSION LPTR(4000),LINE(80)
2000 FORMAT (30A1)
2450 FORMAT (*-2''14,' LINES WRITTEN-'

REWINO TFILE
LIMIT = EOF - 1
DO 90 L = 1,LIMIT
 CALL MEMORY(FETCH,LINE,LPTR(L))
 WRITE (TFILE,2000) LINE
 CONTINUE
 WRITE (OUT,2450) LIMIT
RETURN
END
SUBROUTINE MEMORY(ACTION,LINE,PTR2)

PART OF SLED PACKAGE
HANDLES ALL MEMORY REFERENCES USING DIRECT-ACCESS DISK FILE
CURRENT CAPACITY IS 6000 LINES
REQUIRES AT LEAST 3 DEDICATED CYLINDERS OF DISK SPACE FOR
WORK FILE UNDER CP/CMS ON AN IBM 360/67

COMMON /BLK3/ TFLE
INTEGER WFILE,TFLE,ACTION,STORE,PTR,AVAR,ERRS
INTEGER LINE,LPTR2
DIMENSION LINE(80)
DATA STORE/1/
1000 FORMAT (30A1)

DEFINE WORK FILE
WFILE = 13
DEFINE FILE 13(4000,80,E,AVAR)

CONVERT PTR2 FROM INTEGER*2 TO INTEGER
PTR = PTR2

INITIALIZE READ ERROR COUNTER AND BEGIN
ERRS = 0
IF (ACTION .EQ. STORE) THEN DO
 WRITE (WFILE,PTR,1000) LINE
 ELSE DO
 FETCH
 READ (WFILE,PTR,1000,ERR=99) LINE
 END IF
9, ERRS = ERRS + 1
IF (ERRS .LT. 10) GO TO 5
STOP
FILE: SLED FORTRAN TI

END SUBROUTINE INPUT

C IMPLEMENTS THE INPUT MODE

COMMON /BLK1/ IN, OUT
/BLK2/ LPTR, MAXLIN, EOF
/BLK3/ MFLAG, ERRCT, CURLIN
INTEGER*2 PD, LPTR, BLNK, OUTLIN
INTEGER MAXLIN, EOF, ERRCT, CURLIN, STOE, I, J, IN, OUT, NC
LOGICAL MFLAG
DIMENSION LPTR(4000), OUTLIN(90)
DATA PD/*2, STORE/1, BLNK/* 1

2110 FORMAT (* *)

C IF NO INPUT IN QUEUE, PROMPT USER
C IF (NOT.MFLAG) WRITE (OUT, 2110)
C CALL GETLIN(OUTLIN, NC)

WHILE (NOT.((OUTLIN(1).EQ. PO).AND. (OUTLIN(2).EQ. BL'K)))) DO

IF (NC.GT. 0) THEN DO
C UNLESS IT WAS A NULL LINE
C MAKE ROOM FOR NEW INPUT
IF (CURLIN.LT.EOF) THEN DO
J = EOF - CURLIN
DO 10 J = 1, I
10 LPTR(EOF + 1 - I) = LPTR(EOF - I)
ELSE DO
C CURLIN = EOF
C KEEPS INPUT TEXT CONTIGUOUS
END IF
EOF = EOF + 1
GET A NUMBER FOR NEW LINE FROM STACK
CALL POP(LPTR(CURLIN))
NOW STORE THE NEW LINE
CALL MEMORY(STORE, OUTLIN, LPTR(CURLIN))
CURLIN = CURLIN + 1
END IF
C IF NOTHING IN QUEUE, PROMPT USER
C IF (.NOT.MFLAG) WRITE (OUT, 2110)
C CALL GETLIN(OUTLIN, NC)
END WHILE

RETURN
END

SUBROUTINE GETLIN(OUTLIN, NC)

GETS A LINE FROM TERMINAL; QUEUES UP MULTIPLE LINES

COMMON /BLK1/ IN, OUT
/BLK2/ TCHAR, ERRCT, CURLIN
/BLK3/ MFLAG, NFLAG
INTEGER*2 INLIN, OUTLIN, QUEUE, BLNK, TCHAR
INTEGER IN, OUT, ERRCT, CURLIN, I, J, K, LINELN, NC, NCHARS
LOGICAL MFLAG, NFLAG

19
FILE: SLED FORTRAN TI

DIMENSION INLIN(80), OUTLIN(80), QUEUE(40,1), NCHARS(10)

DATA BLNK,' ', LINELN(80), BLK(80), ENDQ,'/

1010 FORMAT (90A1)

2070 FORMAT ('* -TRUNCATED; ONLY 10 ITEMS PER LINE - *)

C MFLAG SUES TRUE WHEN MULTIPLE INPUT LINES ARE STACKED

IF (.NOT., MFLAG) THEN DO
    READ (IN, 1310, ERR=99, END=98) INLIN
    I = 1
    WHILE ((I.LE.LINELN) AND (INLIN(I).NE.TCHAR)) DO
        OUTLIN(I) = INLIN(I)
        I = I + 1
    END WHILE
    NC = I - 1
    IF (INLIN(I).EQ.TCHAR) MFLAG = .TRUE.
    IF (I.LE.LINELN) THEN DO
        DO 29 J = 1, LINELN
            OUTLIN(K) = BLNK
            CONTINUE
        END IF
    END IF
    WHILE (INLIN(I).EQ.TCHAR) DO
        IF (ENDQ.GE.IO) THEN DO
            WRITE (OUT, 2060)
            INLIN(I) = BLNK
        ELSE DO
            ENDQ = ENDQ + 1
            I = I + 1
            J = J + 1
            NFLAG = .TRUE.
            WHILE ((I.LE.LINELN) AND (INLIN(I).NE.TCHAR)) DO
                QUEUE(J, ENDQ) = INLIN(I)
                IF (INLIN(I).NE.BLNK) NFLAG = .FALSE.
                I = I + 1
            END WHILE
            NC = NCHARS(ENDQ) = J - 1
            IF (I.GT.LINELN) AND NFLAG NCHARS(ENDQ) = 0
            IF (J.LE.LINELN) THEN DO
                DO 30 K = J, LINELN
                    QUEUE(K, ENDQ) = BLNK
                    CONTINUE
                END IF
            END IF
        END IF
        END WHILE
    ELSE DO
        GET LINE FROM QUEUE INSTEAD
        BQ = BQ + 1
        NC = NCHARS(BQ)
        DO 40 1 = 1, LINELN
            OUTLIN(I) = QUEUE(I, BQ)
        CONTINUE
        IF (BQ.EQ.ENDQ) THEN DO
            BQ = 0
            ENDQ = 0
FILE: SLFU  FORTRAN II  NAVAL POSTGRADUATE SCHOOL

MFLAG = *FALSE.

END IF

RETURN

CONTINUE

REWIND IN

CONTINUE

WRITE (OUT,2070)

OUTLIN(1) = OLINK

RETURN

END

SUBROUTINE PUSH(X)

PUSHES A POINTER TO A FREE LINE ONTO THE STACK

COMMON /BLK1/ IN,OUT

1 /BLK2/ STACK,STKPTR

INTEGER STKPTR,IN,OUT

INTEGER*2 STACK

DIMENSION STACK(4000)

2080 FORMAT ("FREE LINE LIST STACK OVERFLOW")

C IF (STKPTR.GT.1) THEN DO

STKPTR = STKPTR - 1

STACK(STKPTR) = X

ELSE DO

STACK OVERFLOW

WRITE (OUT,2080)

END IF

RETURN

END

SUBROUTINE POP(X)

POPS A POINTER TO A FREE LINE FROM THE STACK

COMMON /BLK1/ IN,OUT

1 /BLK2/ LPTR,MAXLIN,EOF

/BLK3/ STACK,STKPTR

INTEGER STKPTR,MAXLIN,EOF,IN,OUT

INTEGER*2 STACK,LPTR,X

DIMENSION STACK(4000),LPTR(4000)

2090 FORMAT ("ALL SYSTEM BUFFERS FULL")

X = STACK(STKPTR)

IF (STKPTR.LT.MAXLIN) THEN DO

STKPTR = STKPTR + 1

ELSE DO

WRITE (OUT,2090)

END IF

RETURN

END
SUBROUTINE CNVRT(STRING, I, J, N)
    CONVERTS CHARACTERS I THROUGH J OF STRING INTO AN INTEGER N
    INTEGER*2 STRING, DIGIT
    INTEGER I, J, N, K, L
    DIMENSION STRING(30), DIGIT(10)
    DATA DIGIT/1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0/
    N = J
    DO 20 K = I, J
        L = 1
        WHILE (STRING(K).NE.DIGIT(L)) DO
            L = L + 1
        END WHILE
        IF (L.LE.10) THEN DO
            N = N + (L-1)*10**(J-K)
        ELSE DO
            N = -99999999
            RETURN
        END IF
        20 CONTINUE
    RETURN
END

SUBROUTINE CONLIN(CL, CLINE, N1, N2, EFLAG)
    FINDS AND INTERPRETS THE LINE NUMBERS CONTAINED ON A COMMAND LINE.
    CHECKS FOR ERRORS.
    INTEGER CL, N1, N2, I, J
    INTEGER*2 CLINE, BLNK, COMMA
    LOGICAL EFLAG
    DIMENSION CLINE(80)
    DATA BLNK/
    /, COMMA/
    EFLAG = .FALSE.
    CONV YOUR FIRST DIGIT
    I = CL
    WHILE ((CLINE(I).NE.BLNK) .AND. (CLINE(I).NE.COMMA)) DO
        I = I + 1
    END WHILE
    IF (I.GE.80) THEN DO
        EFLAG = .TRUE.
    ELSE DO
        CONVERT FIRST NUMBER TO AN INTEGER
        CALL CNVRT(CLINE, I, J-1, N1)
        LOOK FOR SECOND NUMBER
        I = J + 1
        WHILE (CLINE(I).NE.BLNK) DO
            J = J + 1
        END WHILE
        IF (J.GE.80) THEN DO

FILE: SLED FORTRAN T1

NAVAL POSTGRADUATE SCHOOL

EFLAG = .TRUE.
ELSE IF (1 .EQ. J) THEN DO
NO SECOND NUMBER EXISTS.
N2 = N1
ELSE DO
CONVERT SECOND NUMBER.
CALL CVRT (CLINE, I, J-1, N2).
END IF
IF (N1 .GT. N2) EFLAG = .TRUE.
END IF
RETURN
END

SUBROUTINE SEARCH(LINE, STRING, NC, MATCH, MC1)

PART OF SLED PACKAGE.
SEARCHES "LINE" FOR THE FIRST OCCURRENCE OF "STRING".
"MATCH" IS SET TO .TRUE. IF A MATCH IS FOUND.
"NC" IS THE NUMBER OF CHARACTERS IN "STRING" (REQUIRED INPUT).
"MC1" IS AN OUTPUT INDICATING FIRST COL OF MATCH.

INTEGER I, J, L, NC, MC1
INTEGER M LINE, STRING
LOGICAL MATCH
DIMENSION LINE(80), STRING(80)

J = 1
MATCH = .FALSE.
WHILE (.NOT. MATCH) AND (J .LE. 81-NC) DO
WHILE (STRING(I) .NE. LINE(J)) AND (J .LE. 81-NC) DO
J = J + 1
END WHILE
IF (J .LE. 81-NC) THEN DJ =
I = 1
L = J
WHILE (STRING(I+1) .EQ. LINE(L+1)) AND
((L+1) .LT. 80) AND (I+1 .LE. NC) DO
L = L + 1
I = I + 1
END WHILE
IF (I .EQ. NC) THEN DO
MATCH = .TRUE.
ELSE DO
J = J + 1
END IF
END IF
END WHILE
MC1 = J
RETURN
END

23
SLED - FORTRAN IV
FILE: SLED2 FORTRAN II

STANDARD LINE EDITOR—FORTRAN IMPLEMENTATION
SLED VERSION FORTLN NPS MONTGOMERY 9/02/71
PROGRAMMED BY: C. F. TAYLOR, JR., CODE 575

FOR FURTHER INFORMATION SEE NPS TECHNICAL REPORT
NPS 52-03-001 BY L. A. COX, JR.

MAIN PROGRAM:

READS IN COMMANDS AND CALLS THE APPROPRIATE SUBROUTINE
IMPLEMENTS THE "EDIT" MODE

COMMON /BLK/IN,OUT /BLK/MFLAG,ERRCT,CURLIN
INTEGER CL,C2,AC,O,L,M,Q,JK,ST,TV,INLINE,SLNK
DIMENSION INLINE(30)
LOGICAL FLAG,OPENFL,MFLAG
DATA A/"A","C","D","L","M","O","R","V","X","S","T"
/format (** E**)
2000 FORMAT (** -INVALID COMMAND- **Z1)
2020 FORMAT (** -NO TEXT FILE OPENED- **Z1)
2030 FORMAT (** -ONLY ONE FILE CAN BE OPENED PER SESSION IN THIS **VFR
*SIGN OF SLEJ-* )

EDIT MODE--WRITE PROMPT
WRITE (OUT,2000)
CALL GETLINE(INLINE,NC)
CL = INLINE(1)
C2 = INLINE(2)
FLAG = .FALSE.
OPENFL = .FALSE.

MAIN EDIT LOOP
9000 IF (.NOT. IFLAG) GO TO 9001
   IF (.NOT. (.NOT. (OPENFL .OR. (CL.EQ.0) .OR. (CL.EQ.4) .OR. (CL.EQ.
   .EQ.0)) ) ) GO TO 9002
   WRITE (OUT,2000)
   GO TO 9003
9002 CONTINUE
   EULATE CASE (SWITCH) STATEMENT TO PROCESS COMMANDS
   IF (CL .EQ. BLNK) GO TO 200
   IF (CL .NE. L) GO TO 10
   CALL LIST(INLINE)
   GO TO 200
10 CONTINUE
   IF (CL .EQ. S) GO TO 20
   CALL SCREEN(INLINE)
   GO TO 200
20 CONTINUE
   IF (CL .NE. R) .OR. (C2 .NE. S)) GO TO 30
CALL R3(INLINE)
GO TO 200
CONTINUE
30
IF ((C1.NE.W).OR.(C2.NE.L)) GO TO 40
CALL RL(INLINE)
GO TO 200
CONTINUE
40
IF ((C1.NE.A).OR.(C2.NE.L)) GO TO 50
CALL AL(INLINE)
GO TO 200
CONTINUE
50
IF ((C1.NE.D).OR.(C2.NE.S)) GO TO 60
CALL OS(INLINE)
GO TO 200
CONTINUE
60
IF (C1.NE.Q) GO TO 70
CALL QUIT
FLAG = .FALSE.
GO TO 200
CONTINUE
70
IF (C1.NE.V) GO TO 80
CALL VERS
GO TO 200
CONTINUE
80
IF (C1.NE.M) GO TO 90
CALL MENU
GO TO 200
CONTINUE
90
IF (C1.NE.O) GO TO 100
IF (.NOT.(OPENFL)) GO TO 9004
WRITE (OUT,2030)
GO TO 9005
9004
CONTINUE
CALL OPEN
OPENFL = .TRUE.
9005
CONTINUE
GO TO 200
100
CONTINUE
IF ((C1.NE.C).OR.(C2.NE.T)) GO TO 110
CALL CT(INLINE)
GO TO 200
CONTINUE
C
IF PROGRAM GETS HERE, COMMAND IS INVALID
WRITE (OUT,2010) C1,C2
ERRCT = ERRCT + 1
C
END CASE
CONTINUE
200
IF (.NOT.(ERRCT.GE.2)) GO TO 9006
CALL MENU
ERRCT = 0
9006
CONTINUE
4003
CONTINUE
IF (.NOT.((FLAG)) GO TO 9008
C
GET NEXT LINE
FILE: SLE02    FORTRAN II

IF (.NOT., NFLAG) WRITE (OUT, 2000)
CALL VFILE (INLINE, NC)
C1 = INLINE(l)
C2 = INLINE(2)
GO TO 4000
CONTINUE

9008 CONTINUE
GO TO 4000
9001 CONTINUE
STOP
END

BLOCK DATA
COMMON /BLKB/ IN, JUT, /BLK3/ TFILE, BLK4/ TCHAR, BLK5/ MFLAG, EPECT
* CURLIN
INTEGER*2 TCHAR
INTEGER IN, JUT, TFILE, EPECT, CURLIN
LOGICAL MFLAG
DATA IN/5/, JUT/6/, TCHAR/9/, ERECT/0/, MFLAG/.FALSE./, CURLIN/1/, IF
* (LF/2/
END

SUBROUTINE LIST(CLINE)
DISPLAYS TEXT TO THE TERMINAL
COMMON /MLKI/ IN, OUT, /BLK2/ LPTR, MAXLIN, EDF /MLK5/ MFLAG, EPECT,
* CURLIN
INTEGER*2 LNK, LPTR, COMMA, OUTLIN, CLINE
INTEGER IN, JUT, I, J, N1, N2, MAXLIN, EDF, EPECT, CURLIN, FETCH
LOGICAL MFLAG, EFLAG
DIMENSION CLINE(80), OUTLIN(80), LPTR(4000)
DATA BLNK/",", COMMA/",", FETCH/0/
2100 FORMAT ("", INVALID COMMAND")
2110 FORMAT ("", ERROR")
2120 FORMAT ("", EOF")
C IF COL 2 IS BLANK, PRINT CURLIN AND EXIT
IF (.NOT. (CLINE(2) .EQ. BLNK)) GO TO 9010
CALL MEMORY (FETCH, OUTLIN, LPTR(CURLIN))
WRITE (OUT, 2110) CURLIN, OUTLIN
GO TO 4011
9010 CONTINUE
C IF COL 2 IS BLANK, PRINT CURLIN AND EXIT
CALL VMLIN(N2, CLINE, N1, N2, EFLAG)
IF (N1 .EQ. 0) N1 = 1
IF (.NOT. (N2 .EQ. EFLAG)) EFLAG = .TRUE.
IF (.NOT. (N1 .EQ. EFLAG)) EFLAG = .TRUE.
IF (.NOT. (N2 .EQ. EFLAG)) GO TO 9012
EPECT = EPECT + 1
WRITE (OUT, 2110)
GO TO 4013
9012 CONTINUE
EPECT = 0
FILE: SLED2  FORTRAN II  NAVAL POSTGRADUATE SCHOOL

9014 1 = N1
      IF (.NOT. (1 .LE. N2). AND. (1 .LT. EOF)) GO TO 9014
      CALL MEMORY (FETCH, OUTLIN, LPTR(I))
      WRITE (OUT, 2110) I, OUTLIN
      CURLIN = I
      I = I + 1
      GO TO 9014
9015 CONTINUE
9013 CONTINUE
9011 RETURN
C
C SUBROUTINE SCREENCLINE)
C
C DISPLAYS 20 LINES BEGINNING WITH CURLIN OR OTHER SPECIFIED LINE
C
C COMMON /BLK1/ IN,OUT /BLK2/ LPTR,MAXLIN,EJF /BLK5/ MFLAG,ERPCT,
*CURLIN
*INTEGER IN,OUT,MAXLIN,EOF,ERRCT,CURLIN,I,N1,N2,LIMIT,FETCH,N
*INTEGER*2 CLINE,LPTR,OUTLIN,BLNK
*LOGICAL MFLAG,EFLAG
*DIMENSION CLINE(80),OUTLIN(80),LPTR(4000)
*DATA BLNK/*'/'*FETCH/*'
2100 FORMAT (* -INVALID COMMAND- ')
2110 FORMAT (*$'14,'1X,9A1)
2120 FORMAT (* -EOF- ')
C
C EFLAG = 'FALSE.
C
C FIND OUT WHETHER USER SPECIFIED A LINE
C IF (.NOT.(CLINE(N1).NE.BLNK)) GO TO 9014
C      CALL COMLIN12,CLINE,N1,N2,EFLAG
C      IF (N1 .LE. 0) N1 = 1
C      IF (N1 .GE. EOF) EFLAG = 'TRUE.
C      IF (.NOT. EFLAG) CURLIN = N1
9016 CONTINUE
C IF (.NOT.(EFLAG)) GO TO 9018
C      ERRCT = ERRCT + 1
C      WRITE (OUT, 2100)
C      GO TO 9014
9018 CONTINUE
C      LIMIT = MINO(CURLIN+19,EOF-1)
C      DU TO I = CURLIN,LIMIT
C      CALL MEMORY (FETCH, OUTLIN, LPTR(I))
C      WRITE (OUT, 2110) I, OUTLIN
C      CONTINUE
C      CURLIN = LIMIT
C      IF (LIMIT .EQ. EOF-1) WRITE (OUT, 2120)
9019 CONTINUE
C      RETURN
C      END
SUBROUTINE RL(CLINE)

REPLACES CURRENT LINE OR THE SPECIFIED LINE OR LINES WITH
ANY NUMBER OF LINES

COMMON /BLK1/ IN,OUT /BLK2/ LPTR,MAXLNL,EOF /BLK5/ MFLAG,ERRC,
*CURLINE
INTEGER IN,OUT,MAXLIN,EOF,ERRC,CURLIN,N,NL,N2,J,LIMIT,STRE
INTEGER*2 CLINE,LPTR,BLNK
LOGICAL MFLAG,EFLAG
DIMENSION CLINE(BO),LPTR(4000)
DATA BLNK,EOF,STRE/1/
2100 FORMAT (* -INVALID COMMAND-*)
C
N1 = CURLIN
N2 = NL
Determine which line(s) to replace
IF (.NOT.(CLINE(J).NE.BLNK)) GO TO 9020
CALL COMLIN3,CLINE,N1,N2,EFLAG)
IF (N1 .LE. J) EFLAG = .TRUE.
IF (.NOT.EFLAG) CURLIN = N1
9020 CONTINUE
IF (.NOT.(EFLAG)) GO TO 9022
ERRC = ERRC + 1
WRITE (OUT,2100)
GO TO 9023
9022 CONTINUE
ERRC = 0
IF (N2 .GE. EOF) N2 = EOF - 1
C
Remove designated lines
N = N2 - N1 + 1
DO 20 I = 1,N
LIMIT = EOF - 2
CALL PUSHP(LPTR(N1))
DO 10 J = NL,LIMIT
LPTR(J) = (PTR(J+1))
10 CONTINUE
EOF = EOF - 1
20 CONTINUE
C
Now input replacement lines
CALL INPUT
9023 CONTINUE
RETURN
END

SUBROUTINE AL(CLINE)

INPUT TEXT AFTER LINE N

COMMON /BLK1/ IN,OUT /BLK2/ MFLAG,ERRC,CURLIN
FILE: SLEU2 FORTRAN T1
NAVAL POSTGRADUATE SCHOOL

INTEGER*2, BLK, CLINE
INTEGER IN, OUT, J, N, N1, N2, ERRCT, CURLIN
LOGICAL MFLAG, EFLAG
DIMENSION CLINE(40)
COMMON BLK/*'*/
2100 FORMAT (*. INVALID COMMAND-*)

C
C EXTRACT LINE NUMBER FROM COMMAND LINE
C
CALL COMLIN(1, CLINE, N1, N2, EFLAG)
IF (N1 .LT. 0) EFLAG = * TRUE.*
N = N1
IF (.NOT. (EFLAG)) GO TO 9024
ERRCT = ERRCT + 1
WRITE (U1F, 2100)
GO TO 9025
9024 CONTINUE
ERRCT = 0
CURLIN = N + 1
CALL INPUT
9025 CONTINUE
RETURN
END

SUBROUTINE JS(CLINE)

C
C PART OF SLEU PACKAGE
C DISPLAYS ALL LINES CONTAINING THE DESIGNATED STRING, POSSIBLY
C LIMITED TO LINES N THROUGH M.

COMMON /BLK/ IN, OUT /BLK2/ LPTR, MAXLIN, EOF /BLK5/ MFLAG, ERRCT, CUR
LINE
INTEGER IN, OUT, MAXLIN, EOF, ERRCT, CURLIN, N1, N2, NC, FETCH, MC1
INTEGER*2 CLINE, LPTR, BLK, STRING, LINE
LOGICAL MFLAG, EFLAG, MATCH, FOUND
DIMENSION CLINE(40), LLINE(10), LPTR(4000), STRING(40)
DATA BLK/*'*/
2100 FORMAT (*. INVALID COMMAND-*)
2110 FORMAT (*)
2200 FORMAT (*. OLD STRING?*)
2220 FORMAT (*. NO STRING FOUND-*)
C
C DEFAULT VALUES
FOUND = * FALSE.*
EFLAG = * FALSE.*
N1 = 1
N2 = EOF - 1
C
C DETERMINE WHETHER N1, N2 WERE SPECIFIED BY USER
IF (.NOT. (CLINE(3) .NE. BLK)) GO TO 9026
CALL COMLIN(1, CLINE, N1, N2, EFLAG)
IF (N1 .GE. EOF) EFLAG = * TRUE.*
IF (N2 .GE. EOF) N2 = EOF - 1
IF (N1 .LE. 0) N1 = 1
9026 CONTINUE
IF (.NOT. (EFLAG)) GO TO 9028

30
FILE: SLEDZ FORTRAN II

ERRCT = ERRCT + 1
WRITE (OUT,2100)
GO TO 9027
9028 CONTINUE
ERRCT = 0
C
FETCH STRING; ISSUE PROMPT IF NECESSARY
IF (.NOT. MFLAG) WRITE (OUT,2220)
CALL GETLIN (STRING,NC)
IF (.NOT.(NC .LE. 0)) GO TO 9030
ERRLT = ERRCT + 1
WRITE (OUT,2100)
GO TO 9027
9030 CONTINUE
3120 I = N1,N2
CALL MEMORY(FETCH,LINE,LPTR(I))
CALL SEARCH(LINE,STRING,NC,MATCH,MC1)
IF (.NOT.(MATCH)) GO TO 9032
FOUND = .TRUE.
WRITE (OUT,2110) I,LINE
CURLN = I
9032 CONTINUE
20 CONTINUE
IF (.NOT. FOUND) WRITE (OUT,2250)
9031 CONTINUE
9029 CONTINUE
RETURN
END

SUBROUTINE RSICLNE)

PART OF SLED PACKAGE
REPLACES THE FIRST OCCURRENCE OF STRING1 WITH STRING2 ON THE
CURRENT LINE OR WITHIN THE SPECIFIED RANGE OF LINES

COMMON /BLK1/ IN,OUT /BLK2/ LPTR,MAXLNE,EOF /BLK5/ MFLAG,ERRCT,CURL
INTEGER IN,OUT,MAXLNE,EOF,ERRCT,CURLIN,N1,N2,1,J,K,L,M,NI,MC1,MC2
NC2,FETCH,STORE
INTEGER CLINE,LPTR,BLNK,STR1,STR2,L,LINE
DIMENSION LPTR(1400),CLINE(80),STR1(80),STR2(80),LINE(80)
DATA BLNK/1/,FETCH/0/,STORE/I/
LOGICAL MFLAG,FOUND,MATCH,EFLAG
2110 FORMAT (*-*INVALI COMMAND-*)
2110 FORMAT (*-*INVALI COMMAND-*)
2230 FORMAT (* OL/ STRING?*)
2240 FORMAT (* NEW STRING?*)
2250 FORMAT (*-NI STRING FOUND-*)
C
DEFAT Condition
NI = CURLIN
N2 = NI
EFLAG = .FALSE.
C
INTERPRET COMMAND LINE
IF (.NOT.(CLINE(4).NE.BLKNK)) GO TO 9034
CALL CURLIN(3,CLINE,N1,N2,EFLAG)
IF (NI .LE. 0) EFLAG = .TRUE.
FILE: SLED2  FORTRAN TI

IF (IN1 .GE. EOF) EFLAG = TRUE.
IF (IN2 .GE. EOF) NJ = EOF - 1
9034 CONTINUE
IF (.NOT. EFLAG) GO TO 9036
ERACT = ERACT + 1
WRITE (OUT,2100)
GO TO 9037
9036 CONTINUE
ERACT = 0
C READ IN TWO STRINGS: PROMPT IF NECESSARY
IF (.NOT. MFLAG) WRITE (OUT,2230)
CALL GETLINE(STR1,NCL)
IF (NCL .LE. 0) EFLAG = TRUE.
IF (.NOT. MFLAG) WRITE (OUT,2240)
CALL GETLINE(STR2,NC2)
IF (.NOT. EFLAG) GO TO 9038
ERACT = ERACT + 1
  WRITE (OUT,2100)
RETURN
9038 CONTINUE
C NOW FIND STRING1
FOUND = .FALSE.
DO 50 K = NJ,N2
CALL MEMORY(FETCH,LINEL,LPTR(K))
CALL SEARCH(LINE,STRING1,NCL,MATCH,MCL)
IF (.NOT. MATCH) GO TO 9040
C NOW MAKE SUBSTITUTION
J = MCL
FOUND = .TRUE.
C DELETE STRING1
DO 20 I = 1,NCL
DO 10 M = J-79
LINE(M) = LINE(M+1)
10 CONTINUE
20 CONTINUE
C NOW MAKE ROOM FOR STRING2
IF (.NOT. (NC2 .GT. 0)) GO TO 9042
DO 40 L = 1,NC2
M = 81 - J
DO 30 I = 2,M
LINE(82-I) = LINE(81-I)
30 CONTINUE
40 CONTINUE
C NOW INSERT NEW STRING
DO 45 I = 1,NC2
LINE(81+I-1) = STR2(I)
45 CONTINUE
C STORE REVISED LINE
CALL MEMORY(STORE,LINEL,LPTR(K))
C DISPLAY REVISED LINE
WRITE (OUT,2110) K,LINE
C REMOVE "C" IN CC 1 IF ABOVE LINE TO ENABLE
C DISPLAY OF EACH LINE IN WHICH A STRING HAS BEEN
C REPLACED

32
FILE: SLE02  FORTRAN T1

NAVAL POSTGRADUATE SCHOOL

CURLIN = K

9040 CONTINUE
50 CONTINUE
IF (.NOT. NIT. FOUND) WRITE (OUT,2250)
9037 CONTINUE
RETURN
END
SUBROUTINE CT

C PART OF SLE) PACKAGE
C CHANGES THE MESSAGE TERMINATOR TO ANY VALID CHARACTER
COMMON /BLK1/ INT,OUT /BLK4/ TCHAR /BLK5/ MFLAG,ERRCT,CURLIN
INTEGER IN,OUT,ERRCT,CURLIN,NC
INTEGER *2 TCHAR,INLIN,BLNK
LOGICAL MFLAG;
DIMENSION INLIN(80)
DATA BLNK/* '
2100 FORMAT (' -INVALID COMMAND-' )
2200 FORMAT (' TERMINATOR??')

C IF (.NOT. (.NOT. MFLAG)) GO TO 9044
C ISSUE PROMPT
WRITE (OUT,2200)
9044 CONTINUE
CALL GETLIN(INLIN,NC)
IF (.NOT. (.NOT. NC. EQ. 0).OR.(INLIN(1).EQ.BLNK)) ) GO TO 9046
WRITE (OUT,2100)
GO TO 9047
9046 CONTINUE
TCHAR = INLIN(1)
9047 CONTINUE
RETURN
END
SUBROUTINE MENU

C PART OF SLEO PACKAGE
C PROVIDES USER WITH A SUMMARY OF AVAILABLE COMMANDS
AND THEIR FORMATS.
C
COMMON /BLK1/ IN,OUT
INTEGER IN,OUT
200 FORMAT (' SLED COMMAND SUMMARY:/' ,LINE/TEXT INSERT STRING
* REPLACEMENT/3X,ALN',T10,'INSERT CHAR FTER LINE N',T10,'NO',T10,'RSN',T10,'PLACE STRING/3X,RLN',T10,'REPLACE STRING LINE N OR
** T40,'RSN3POS',T50,'**' WITH 'OR IN' / 3X,'RLN',T40,'T15,'LINES N TO
* T1,,'L40,'T60,'LINES N TO T52,' 'INDICATED LINES' /'' OUTPUT COMMANDS',
* T38,'STRING SEARCH/3X,T10,'DISPLAY CURRENT LINE N',T40,'USN
** T51,'T48,'DISPLAY LINES 3X,'L4',T10,'OR LINE N',T10,'WITH <S>
* T49,'PH',3X,'IN',T10,'LINES V THRU 4',T47,'USN4',T49,'T49',T50
* READ <S> LINES'
201 FORMAT (3X,'ST',T10,'<S>HOW A ', 'SCREEN OF LINES',T30,'<N-1 CONTAIN
* LINING "P"/3X,SN',T10,'SHOW A SCREEN FROM LINE N',T10,'CONT?',FLG C
* COMMANDS/3X.'4',T10,'SHOW COMMAND <F>NU (THIS)',T10,'<')
FILE: SLE02 FORTRAN T1

OPEN A FILE USE /3X*V*TIO,*SHOW, *VERSION INFORMATION, *TD, *CREA
TE A FILE FOR EDITING/*40,*CT, CHANGE THE LOGICAL/*5X,*TD,<3>U
*IT THE FILE TYPE */<RET>-"T/>4J,*T4J,*MESSAGE *TERMIN, *AJ4K)
WRITE ((OUT,220))
WRITE ((OUT,221))
RETURN
END
SUBROUTINE VFPS

FILE FOR A/RTING INTO THE LOGICAL/IX'
FILE F01 T TYPE: "lJ<Rc-T>"'T4J, "MFSS4GE TFVll', '.AtR
WRITE ((JUT92L)) END SUBROUTINE VFPS

PART OF SLEJ PACKAGE

COMMON /BLK/ IN,OUT
INTEGER IN,OUT
220 FORMAT (*) SLEJ VERSION 30T1.1 MPS WRITEKEY 30D4.1/* LOCAL EXPER
* 1 (S. TAYLOR 4006-646-2691 0900-1700 "ST/POT")/* LIME FDELETE KEY
* IS < > (ASCII) OR <CENT SIGN> ' "ESQCIT/' CHARACTER DELETE KEY
* EDITOR LOGICAL MESSAGE TERMINATORS ARE */ (1)
*<RETURN> AND (2) < > */ AND CAN BE CHANGED TO ANY STANDAPP
* FORTRAN CHARACTER */ ALL INPUT IS TRANSLATED TO UPPER CASE */
* THE FOLLOWING DEVIATIONS FROM SLED STANDARD ARE REQUIRED/*

221 FORMAT ("1") THE UNIVERSAL ENTRY COMMAND "SLED" INVOKES INSTRIUC
*TIONS */ FOR A NON-STANDARD ENTRY: "SLED1 <FILENAME> <FILEL
*YPE> */ (2) ONLY ONE FILE PER SESSION CAN BE "OPENED */ (1)
*3) MAXIMUM FILESIZE IS 4000 LINES */ (4) THE USER IS ASKED TO
*INDICATE WHETHER HE IS EDITING A */ A NEW FILE IN ORDER TO PR
*EVENT A DISK READ ERROR IN FORTRAN 4. */ (5) WHEN <RETURN> IS USED
*AS A LOGICAL MESSAGE */ TERMINATOR, THE LINE OR STRING IS
*PADDED WITH BLANKS ON */ THE RIGHT. THIS AFFECTS THE RS F
*FUNCTION ONLY */

222 FORMAT ("6") ONLY THE FIRST OCCURRENCE OF A STRING IN EACH LINE
*/ IS REPLACED TO PERMIT FREE SUBSTITUTIONS OF BLANKS*/
WRITE ((OUT,220))
WRITE ((OUT,221))
RETURN
END
SUBROUTINE OPEN

OPENS TEXT FILE AND WORKSPACE FILE
READS TEXT FILE INTO WORKSPACE IF IT ALREADY EXISTS
INITIALIZES POINTERS ETC.

COMMON /BLK/ IN,OUT /BLK2/ LPTR,MAXLIN,EOF /BLK3/ TFILE /BLK5/ MF
*LAG,ERCT,CUHLIN /BLK6/ STACK,STKPR
INTEGER IN,OUT,TFILE,LN,ERRC,CUHLIN,STORE,MAXLIN,EOF,STKPTR,1,
*NL
*INTEGER (*2 LPTR,STACK,FNAME,INLIN,YES,NO,REPLY,BLINE
LOGICAL MFLAG
DIMENSION LPTR(4000),BLINE(90),STACK(4000),FNAME(90),INLIN(90)
DATA YES/"Y"/,NO/"N";/*STORE/1/
1000 FORMAT (80A1)
2060 FORMAT (*-"I",LINES IN FILE: "80A1)
2070 FORMAT (*-CREATING FILE: "80A1)
2400 FORMAT (* FILENAME?"
2410 FORMAT (* IS THIS A NEW FILE?>"
FILE: SLED2 FORTRAN T1
NAVAL POSTGRADUATE SCHOOL

2420 FORMAT (' - MAX CAPACITY 4000 LINES EXCEEDED-')

C

INITIALIZE
MAXLIN = 4000
CUPLIN = 1

C

READ IN FILENAME (COSMETIC)
IF (.NOT. MFLAG) WRITE (OUT, 2400)
CALL GETLINE(NAME, NC)

C

ASK WHETHER IT IS A NEW FILE (TO PREVENT FORTRAN READ ERROR)
5 WRITE (OUT, 2410)
READ (IN, 1000) LINL
REPLY = 'LINL(1)
IF ( (REPLY .NE. YES), AND, (REPLY .NE. NO) ) GO TO 5
IF ( .NOT. (REPLY .EQ. YES) ) GO TO 9048
WRITE (OUT, 2050) FName
EOF = 1
STKPTR = 1
ACTIVATE FILE WITH AN ACCESS
LPTK(1) = 1
CALL MEMORY(STORE, BLINE, LPTK(1))
GO TO 9049

9048 CONTINUE

9050 IF (.NOT. (LINE .LE. MAXLIN)) GO TO 9051
READ (TFILE, 1000, END=10) INLIN
LPTK(LINE+1) = LINE + 1
CALL MEMORY(STORE, INLIN, LPTK(LINE+1))
LINE = LINE + 1
GO TO 9050

9051 CONTINUE
10 CONTINUE
IF (.NOT. (LINE .GE. MAXLIN)) GO TO 9052
WRITE (OUT, 2420)
STOP

9052 CONTINUE
STKPTR = LINE + 1
EOF = STKPTR

9049 CONTINUE
DO 20 I = 1, MAXLIN
STKPT(I) = 1
20 CONTINUE
RETURN

END SUBROUTINE QUIT

C

PART OF THE SLED PACKAGE
C CLOSES OUT THE WORK FILE AND WRITES THE NEW OR UPDATED
C TEXT FILE

C

COMMON /BLK2/ LPTK, MAXLIN, EOF, /BLK3/ TFILE /BLK1/ IN, OUT
INTEGER MAXLIN, IN, OUT, EOF, TFILE, LIMIT
INTEGER*2 LPART
FILE: SLEN2  FORTRAN TI

DIMENSION LPTR(4000), LINE(80)
2000 FORMAT (40AI)
2450 FORMAT (4,14,4 LINES WRITTEN-1)

C REWIND TFILE
LIMIT = EFI - 1
DO 90 L = 1, LIMIT
CALL MEMORY(FETCH, LINE, LPTR(L))
WRITE (TFILE, 2000) LINE
90 CONTINUE
WRITE (OUT, 2450) LIMIT
RETURN
END

SUBROUTINE MEMORY(ACTION, LINE, PTR2)

C PART OF SLED PACKAGE
C HANDLES ALL MEMORY REFERENCES USING DIRECT-ACCESS DISK FILE
C CURRENT CAPACITY IS 4000 LINES
C REQUIRES AT LEAST 3 DIRECT-ACCESS CYLINDERS OF DISK SPACE FOR
C WORK FILE UNDER CP/CMS ON AN IBM 360/67
C
COMMON /BLK3/ TFILE
INTEGER WFILE, TFILE, ACTION, STORE, PTR, AVAR, ERRS
INTEGER PTR2
DIMENSION LINE(80)
DATA STORE/1/
1000 FORMAT (40AI)
C
C DEFINE WORK FILE
WFILE = 13
DEFINE FILE 13(4000, 80, E, AVAR)

C CONVERT PTR2 FROM INTEGER*2 TO INTEGER
PTR = PTR2
C
C INITIALIZE READ ERROR COUNTER AND BEGIN
ERRS = 0
IF (.NOT. (ACTION .EQ. STORE)) GO TO 9054
WRITE (WFILE, PTR, 1000) LINE
GO TO 9055
9054 CONTINUE
C FETCH
5 READ (WFILE, PTR, 1000, ERR=99) LINE
9055 CONTINUE
RETURN
99 ERRS = ERRS + 1
IF (ERRS .LT. 10) GO TO 5
STOP
END

SUBROUTINE INPUT

C IMPLEMENTS THE INPUT MODE

COMMON /BLK1/ IN, OUT /BLK2/ LPTR, MAXLIN, EOF /BLKS/ MFLAG, ERRCT, CUR *
LIN
INTEGER*2 PD, LPTR, BLNK, OUTLIN
INTEGER MAXLIN, EOF, ERRCT, CURLIN, STORE, I, J, IN, OUT, NC

36
FILE: SLED2  FORTRAN T1

LOGICAL AFLAG
DIMENSION LPR(4030), OUTLIN(90)
DATA P(7), STRP(1), BLNK(1), /'
2110 FORMAT (I, [''] I)
C IF NO INPUT IN QUEUE, PROMPT USER
IF (.NOT. AFLAG) WRITE (OUT, 2110)
CALL GETL((OUTLIN, NC)
9056 IF (.NOT. (.NOT. (OUTLIN(1), EOF), EOF)) AND (OUTLIN(2), EOF, BLNK)) GO TO 9057
IF (.NOT. IN, EOF) GO TO 9058
C IF LESS IT WAS A NULL LINE
MAKE ROOM FOR NEW INPUT
IF (.NOT. (CURLIN.LT. EOF)) GO TO 9060
J = EOF - CURLIN
I = EOF - 1
LPR(EOF + 1 - I) = LPR(EOF - 1)
CONTINUE
GO TO 9061
9060 CONTINUE
C CURLIN = EOF
C KEEPS INPUT TEXT CONTIGUOUS
9061 CONTINUE
C GET A NUMBER FOR NEW LINE FROM STACK
C NOW STORE THE NEW LINE
CALL MEMORY (STORE, OUTLIN, LPR(CURLIN))
CURLIN = CURLIN + 1
9058 CONTINUE
C IF NOTHING IN QUEUE, PROMPT USER
IF (.NOT. AFLAG) WRITE (OUT, 2110)
CALL GETLINE (OUTLIN, NC)
GO TO 9056
9057 CONTINUE
RETURN
END

SUBROUTINE GETLINE (OUTLIN, NC)
C GETS A LINE FROM TERMINAL; QUEUES UP MULTIPLE LINES
C COMMON /BLK1/ IN, OUT /BLK4/ TCHART /BLK5/ MFLAG, ERRCT, CURLIN
INTEGER *2 INLIN, OUTLIN, QUEUE, BLNK, TCHART
INTEGER IN, OUT, ERRCT, CURLIN, J, K, LINEL, BN, END, NC, NCHARS
LOGICAL MFLAG, NFLAG
DIMENSION INLIN(80), OUTLIN(80), QUEUE(80, 10), NCHARS(10)
DATA BLNK/1, 3, 10, 40, 0, 0, 0, 0, 0, 0/ 
1010 FORMAT (90A)
2050 FORMAT (9, ' TRUNCATED; ONLY 10 ITEMS PER LINE-')
2070 FORMAT (9, ' -ILLEGAL CHARACTER OR BLANK COMMAND-')
C MFLAG GUES TRUE WHEN MULTIPLE INPUT LINES ARE STACKED
IF (.NOT. (.NOT. AFLAG)) GO TO 9062
READ (IN, 1010), ERR=99, END=98) INLIN

37
FILE: SLEUZ
FORTRAN TI
NAVAL POSTGRADUATE SCHOOL

9064 IF (.NOT. ((I.LE.LINELN).AND.(INLIN(I).NE.TCHAR)) ) GO TO 9065
    OUTLIN(I) = INLIN(I)
    I = I + 1
    GO TO 9064

9065 CONTINUE
    NC = I - 1
    IF (INLIN(I).EQ.TCHAR) MFLAG = .TRUE.
    IF (.NOT.(I.LE.LINELN)) GO TO 9066
    DO 20 K = I,LINELN
        OUTLIN(K) = BLNK
    20 CONTINUE
    CONTINUE

9066 CONTINUE

9067 IF (.NOT. (INLIN(I).EQ.TCHAR)) GO TO 9069
    IF (.NOT.(ENDQ.GE.10)) GO TO 9070
    INLIN(I) = BLNK
    GO TO 9071

9070 CONTINUE
    ENDQ = ENDQ + 1
    I = I + 1
    J = 1
    NFLAG = .TRUE.

9072 IF (.NOT. ((I.LE.LINELN).AND.(INLIN(I).NE.TCHAR)) ) GO TO 9073

9073 WRITE (OUT.2060)
    INLIN(I) = BLNK
    NFLAG = .FALSE.
    J = J + 1
    GO TO 9072

9074 CONTINUE

9075 NC = CHAR(ENQ) = 0
    IF (BQ.EQ.ENQ) GO TO 9016
    IF (.NOT.(BQ.EQ.ENDQ)) CONTINUE

9076 CONTINUE

9063 CONTINUE

C GET LINE FROM QUEUE INSTEAD

BQ = BQ + 1
    NC = NCHARS(BQ)
    DO 40 I = 1,LINELN
        OUTLIN(I) = QUEUE(I,BQ)
    40 CONTINUE
    IF (.NOT.(BQ.EQ.ENDQ)) GO TO 9076

9076 CONTINUE

9076 CONTINUE
FILE: SLFO?2  FORTRAN TL

RETURN
98 CONTINUE
99 CONTINUE
WRITE (OUT,2070)
OUTLIN(I) = BLNK
RETURN
END

SUBROUTINE PUSH(X)
C
PUSHES A POINTER TO A FREE LINE INTO THE STACK
C
COMMON /BLK1/ IN,OUT /RLK6/ STACK,STKPTR
INTEGER STKPTR,IN,OUT
INTEGER*2 STACK,X
DIMENSION STACK(4000)
2030 FORMAT (1 -FREE LINE LIST STACK OVERFLOW-')
C
IF (.NOT.(STKPTR .GT. 1)) GO TO 9078
STKPTR = STKPTR - 1
STACK(STKPTR) = X
GO TO 9079
9078 CONTINUE
C
STACK OVERFLOW
WRITE (OUT,2080)
9079 CONTINUE
RETURN
END

SUBROUTINE POP(X)
C
POPS A POINTER TO A FREE LINE FROM THE STACK
C
COMMON /BLK1/ IN,OUT /BLK2/ LPTR,MAXLIN,EOF /RLK6/ STACK,STKPTR
INTEGER STKPTR,MALXIN,EOF,IN,OUT
INTEGER*2 STACK,LPTR,X
DIMENSION STACK(4000),LPR(4000)
2090 FORMAT (1 -ALL SYSTEM BUFFERS FULL-')
C
IF (.NOT.(STKPTR .LT. MAXLIN)) GO TO 9080
STKPTR = STKPTR + 1
GO TO 9081
9080 CONTINUE
WRITE (OUT,2090)
9081 CONTINUE
RETURN
END

SUBROUTINE CNVRT(STRING,I,J,N)
C
CONVERTS CHARACTERS I THROUGH J OF STRING INTO AN INTEGER N
FILE: SLED2  FORTRAN I1

C
INTEGER*2 STRING, DIGIT
INTEGER I, J, N, K, L
DIMENSION STRING(80), DIGIT(10)
DATA DIGIT/ ,9,1,2,3,4,5,6,7,8,9/, N = 0
DO 20 K = 1, L
L = 1
9082 IF (.NOT. (STRING(K).NE.DIGIT(L))) GO TO 9083
L = L + 1
9083 CONTINUE
GO TO 9082
9084 CONTINUE
9085 CONTINUE
RETURN
20 CONTINUE
RETURN
END

SUBROUTINE COMLIN(CL,CLINE,N1,N2,EFLAG)
C
C FINDS AND INTERPRETS THE LINE NUMBERS CONTAINED ON A
C COMMAND LINE. CHECKS FOR ERRORS.
C
INTEGER CL,N1,N2,I,J
INTEGER*2 CLINE,BLNK,COMMA
LOGICAL EFLAG
DIMENSION CLINE(80)
DATA BLNK/*",COMMA/*,
EFLAG = .FALSE.
C
C FIND FIRST DIGIT
I = CL
9086 IF (.NOT. ((CLINE(J).NE.BLNK).AND.(CLINE(J).NE.COMMA))) GO TO 9091
J = J + 1
GO TO 9086
9087 CONTINUE
GO TO 9088
9088 CONTINUE
C
C CONVERT FIRST NUMBER TO AN INTEGER
CALL CNVRT(CLINE,I,J-1,N1)
C
C LOOK FOR SECOND NUMBER
I = J + 1
J = 1
9090 IF (.NOT. (CLINE(J).NE.BLNK)) GO TO 9091
J = J + 1
GO TO 9090
CONTINUE IF (.NOT.(J.GE.90)) GO TO 9092
EFLAG = .TRUE.
GO TO 9093
9092 CONTINUE IF (.NOT.(I.EQ.J)) GO TO 9094
C NO SECOND NUMBER EXISTS
N2 = N1
GO TO 9095
9094 CONTINUE CONVEXT SECOND NUMBER
CALL CNVRT (CLINE, I, J-L, N2)
9095 CONTINUE IF (IN1 GT N2) EFLAG = .TRUE.
9093 CONTINUE
9099 CONTINUE RETURN
END
SUBROUTINE SEARCH (LINE, STRING, NC, MATCH, MC)
C PART OF SLED PACKAGE
SEARCHES 'LINE' FOR THE FIRST OCCURRENCE OF 'STRING'.
'MATCH' IS SET TO 'TRUE.' IF A MATCH IS FOUND.
'NC' IS THE NUMBER OF CHARACTERS IN 'STRING' (REQUIRED INPUT)
'MC' IS AN OUTPUT INDICATING FIRST COL (IF MATCH
INTEGER I, J, L, NC, MC
INTEGER*2 LINE, STRING
LOGICAL MATCH
DIMENSION LINE(80), STRING(80)
C J = 1
MATCH = .FALSE.
9096 IF (.NOT. ((.NOT.MATCH) .AND. (J.LE.81-NC))) GO TO 9097
9098 IF (.NOT. ((STRING(I+1).EQ.LINE(J)) .AND. (J.LE.81-NC))) GO TO 9094
*9999 CONTINUE IF (.NOT.(J.LE.81-NC)) GO TO 9100
I = 1
L = J
9102 IF (.NOT. ((STRING(I+1).EQ.LINE(L+1)) .AND. ((L+1).LT.80) .AND.
*0.((I+1).LE.NC))) GO TO 9103
L = L + 1
I = I + 1
GO TO 9102
9103 CONTINUE IF (.NOT.(I.EQ.NC)) GO TO 9104
MATCH = .TRUE.
GO TO 9105
9104 CONTINUE
9105 CONTINUE
9100 CONTINUE
FILE: SLED  FORTRAN T1

GO TO 9096

9097 CONTINUE

MC = J

RETURN

END
The purpose of this programming project was to implement a simple text editor to run under standard PASCAL as defined in Wirth [1972]. The specifications for the program are fairly extensive and seek to define the program in specific enough terms to ensure portability. In designing and implementing this program, the following two goals were utilized:

a. PORTABILITY. The finished program should be capable of running under any implementation of standard PASCAL.
b. STANDARDIZATION. The finished program should abide by the detailed specifications provided for the user interaction with any implementation dependent features fully documented to facilitate use by both inexperienced and experienced users.

SYSTEM DESIGN

The overall design for the text editor was heavily influenced by the strict requirements of the specification document. This specification delineated the commands that were to be implemented and their format. The primary design task involved creating an efficient system which included the required commands in an implementation independent program.

FILE MANAGEMENT. The primary purpose of any text editor is to create and modify text (character) files in an interactive manner. This problem can be separated into several functional areas. The first of these is file manipulation and management. In as much as PASCAL was designed primarily as a pedagogical aid, the language lacks extensive input/output operations. This in turn allows the various implementations to define these operations. Since this would necessarily result in operations which were not portable, it was decided to design the program to meet the requirement that the user be able to access external text files from within the program and that the program not fail if the user attempted to access a non existant file.

The Berkeley Pascal implementation for the UNIX operating system (which was used for this effort) defines three types of files, in addition to the standard input and output files. The first of these concerns explicit naming. In this case, the file name is placed in the program heading and acts as a passing "parameter" from the UNIX operating system into the program. The file so named must exist as a UNIX...
file.

The second type is an implicitly defined file. These files are declared in variable declaration sections of program blocks and have scope in the same manner as variables. When a block is entered, the file is created with the UNIX filename tmp.x where x is an integer representing the chronological order in which the file is used. When the block is exited, the file is destroyed.

The third type of file uses a dummy file name convention. The file name is declared in the variable declaration section but can be equivalenced to an existing file by the system functions reset and rewrite. These functions create a UNIX path between the dummy file name and the actual UNIX name, which may be supplied during execution. In the case of rewrite, if no UNIX file exists, it is created.

The dummy file name seemed ideally suited to the program requirement that the user be able to create and access files at will. However, it is necessary to know whether the file which is to be opened has been previously created. Since it is impossible for the program to access the UNIX user's directory to determine if files exist, it was decided to create a SLED directory containing the file names and sizes of all files created in the SLED environment. In the event that the user desired to edit a file which existed in his UNIX directory but not in the SLED directory, the SLED directory could be edited to include this filename. To implement this feature, it was decided to explicitly name this directory <directory> as a UNIX file. Any additional files needed as temporary storage locations to be utilized while a file was being edited could then be implemented as implicitly named files, since their existence is not important to the user (unless the amount of file space allocated to the user by the operating system is limited).

TEXT MANIPULATION. As a text editor, SLED has to carry out two basic functions. The editor must insert and delete lines of text in the file, and must search lines for pattern matching and replacement. These functions are related and are the dominant factor in the choice of an effective data structure.

PASCAL has several data structure constructors in the language which can be used to fulfill these tasks. These include the linear array, record, and pointer. The salient features of these constructors in regards to the tasks involved will now be discussed.

The PASCAL array is similar to arrays in other languages with
the exception that the elements of the array may be complex data types such as records. Since the array must be statically defined, the storage can be highly efficient. Lines of text can be stored as character arrays. This has the advantage that locating a given line or character can be accomplished by simply subscripting a variable. In addition, overhead is at a minimum since no pointers, links or other devices are required. The array suffers from the disadvantage that insertions and deletions are expensive as they require copying on the average half of the array. Furthermore, since the array must be defined statically, it is likely that much of the array will be empty at any one time.

The record data structure is similar to the array except that the elements of the record need not be of the same data type. By itself, the record offers no advantages as compared to the array. However, the record fields can be used to act as pointers or links to other records thus allowing the creation of lists or trees. Trees offer the advantage that sorting, inserting, and deleting can be carried out quite efficiently. In addition, searching is easier and more efficient than with linked lists (although not as efficiently as an array). The major disadvantage of the tree is the large amount of overhead required since each interior node must contain a link to each descendant. The linked list can be considered as a compromise of the tree structure. Insertions and deletions are still efficient, but searching requires following a string of pointers through the list. Since only one pointer is required for each node, the amount of overhead is approximately halved compared to the tree.

The pointer type represents a method of dynamic allocation of records to a linked list or tree structure. This offers the advantage that space need not be allocated until actually required. Unfortunately, the methods for allocating and deallocating memory space via pointers is poorly defined in the language, resulting in implementation dependent designs. In particular, standard PASCAL allows pointers to records to be destroyed resulting in the creation of garbage. No garbage collection is carried out by the language to recover this memory. Therefore, a common technique is to place unused records into a "free" list. This, however, defeats the idea of dynamic allocation.

Based on these considerations, it was decided that lines within the text file would be treated as a linked list. This would facilitate the line append and line replace operations. The overhead for this structure would then be one pointer per line, which was not considered excessive. Rather than depend on the pointer type to create and manage list elements, it was decided that the list would consist of an array of line records, with each record consisting of a pointer to the next line and the contents of the line.

In representing the characters of a given line, it was necessary to decide between using another linked list for elements of a line or a character array. Again, the linked list would make character insertions
and deletions efficient. Since this would require a pointer for each character, about half of the memory space allocated would be overhead. A possible compromise would be to have each list node consist of several characters. This would create difficulties in insertion and deletion and would require a complicated algorithm to implement. It was decided to represent the line as a character array. This had the benefit of making the pattern matching algorithm easier to implement as well as reducing overhead.

PARAMETERIZATION. To allow for adaptations of the program to other systems, the parameters for the data structure are defined in a constant declaration block in the main program. This allows implementations to scale the size of the data structure to the amount of memory available. To further enhance portability, it was decided to localize the input/output procedures in separate routines which could be replaced when implementing the system on other machines.

IMPLEMENTATION

The linked line list was implemented as an array of line records named <buff>. Each line record consists of an integer pointer to the array element (record) of the next line and a 120 character packed array. This size allows the creation of a line which will cover the linesize of most standard output devices. In addition, a separate record, <head>, serves as a pointer into the line list and contains the number of the first line presently in the buffer.

GENERAL STRUCTURE OF SLED. The editor basically has five categories of text processing procedures. The first of these are the control commands. There are three types of control commands: a change of logical message terminator so that the user may select the symbol or character he desires to indicate an end of a line or command, a command to exit the editor mode which will write the text file to the user's file as well as terminate the program, and a command to open a file for the user, either a new file or an existing file from his directory and close any previously opened file. As discussed in the section on design, this was implemented as a separate SLED directory. The contents of the directory consist of the UNIX file name along the total number of text lines in the file. In addition to this, a scratch file is maintained to allow updating of the directory contents.

To implement the logical terminator, it was decided to limit the terminators to printable characters. This allowed the terminator to serve as an end of line signal in the text buffer, eliminating the need for creating a separate list of line lengths or employing some other line length algorithm.

The second group of commands consists of those commands
concerned with output of the text file to the CRT screen. These commands are divided into those which display a specific portion of text and those which display large blocks of text. These commands, called output commands, consist of routines to display the current line, a specific line or a designated number of lines. The user optionally selects a from-line/to-line pair of numbers for display and defaults to the current line (defined as the last line displayed or last line operated upon). Another output command screens a large block of text for the user by using a from-line input. These display commands allow the user to edit large portions of his text. Two other commands, which do not process or handle text, and are considered output commands are a procedure to display a command menu to assist the user with SLED procedures and a display of the version listing for a more sophisticated user.

The third type of commands are those which handle insertion of text into the file. The line insertion commands cause the editor to enter the insert mode (all other procedures are in the edit mode). These commands allow insertion of new lines into the text as well as replacing a specific line or a group of lines in the text. In this way, the user can create or destroy portions of his text file by linking the new lines into the buffer list.

A fourth command searches the text for a particular character or string of characters and displays them to the user. Closely related to this is the command to replace these characters and strings in the body of the text, either in a specific line, a group of lines or throughout the entire text. These two types of commands along with the line insertion commands form the basis of the text processing procedures, while the output and control commands form the basis of the text handling procedures. To implement the pattern matching routine the Knuth, Morris, Pratt algorithm (as discussed in Knuth and elsewhere) was utilized. This algorithm uses the concept of a finite automation to determine how far to advance the pattern along the target line in the event of a mismatch between the pattern and its target. This is done by creating a table of edges which represent failure in the automaton. As an example, if the pattern consists of three identical letters and the first two match but the third one does not, instead of advancing the pattern by one position relative to the target the pattern can be moved three places since the first character cannot possibly match the third character of the target. The next table determines how far to advance the pattern if matching fails with the ith character of the pattern. In this way, no back-tracking in the target is required and the algorithm is $O(n)$. The diagram attached as Figure B1.1 shows the general scope of SLED as defined in the specification document. It basically shows the editor commands required by the system grouped into the five primary command areas; output, control, line insertion, string search and string replacement. The diagram also shows the different editor command modes.

SPECIFIC STRUCTURE OF SLED. Based on the general structure of SLED as proposed by the problem specification and as diagramed above,
the programming team made a detailed study of the basic routines needed, how these routines would interact and the data structures and file handling procedures that would be needed. The Figure B1.2 is a schematic of the implementation of SLED as described in this documentation. In implementing the SLED program, a top-down methodology was utilized in defining the program processes needed to meet the requirements stated above.

There are four general sections or levels to the program. The first level acts as a traffic controller for the entire program. It reads each of the user input commands and branches to the appropriate subroutine. The main program functions as this first level and screens the commands, eliminating the incorrect ones and processing the properly entered ones. It is the framework in which SLED performs its functions.

At this same level, the change terminator is located primarily because it does not handle the user data except to place the new terminator symbol in the text file (see procedure Changeterm documentation). It is properly classified as a special procedure in program control rather than text processing or text handling.

The second level of the program contains the bulk of the text processing procedures and consequently, the bulk of the coding. One subgroup contains the procedures which display the lines of text from the user's file. In many ways, it is similar to the output section outlined in the original evaluation. In addition to the screen line and display line procedures, we have included the string search procedure in this subgroup because it functions in a similar manner in that it is involved in the display of strings within the lines of text. This subgroup interfaces with a major subgroup of the third level, namely the commands which read the user's line number designations, translate them, and fetch the lines from the text file.

The second major subgroup of this level consists of the commands which handle the majority of the text insertions, deletions and string processing. These include appending lines, replacing lines and replacing or changing strings. These commands were grouped together due to their similarity of function and commonality of coding. They each have major subprocedures at this level and interface with the primary text handling procedures at the fourth level.

Also found at the second level are two other groups of procedures which are of less importance than the text processing procedures but are useful and necessary segments of the SLED program. The first of these are the control commands which open and close the file. They are obviously required file handling procedures and perform the functions normally expected of a text editor. They interface with key implementation dependent procedures at the third level.

The last major subgroup of the second level is the required
command menu and version document. The call to these procedures is rather simple and uncomplicated. Rather than place the documents in the PASCAL program, we decided to employ the existing UNIX directory to hold them. These files are explicitly named in the program as <menu> and <version>. The file contains the current command menu as well as the version and any changes to either of these to facilitate assisting the user can be made quickly and efficiently.

The third level of programming of SLED contains two subgroups, each of which are subroutines for major procedures in level two. The first subgroup are the routines which transform the user line number requests into from-line/to-line pairs and fetch the appropriate lines from the user's text. They are text handling commands and are part of the output section in the original specification.

The second subgroup interacts with the open and close file routines and are key text handling procedures. They are dependent upon the implementation of PASCAL in use on the computer system. The procedures utilize a directory file which, like the menu and version, is located in the UNIX directory.

The fourth level of SLED contains the key text handling commands to move data in and out of the buffer when required. All of the major procedures of SLED call the read buffer and write buffer routines to move through the user's file. These procedures are also implementation dependent.

A minor procedure of the program is also found at this level which causes the command menu to be printed when the user makes two consecutive errors, a requirement of the SLED specification.

TESTING AND EVALUATION

The constraints of time prohibited an exhaustive and thorough evaluation of SLED. There has, however, been extensive and continuous testing of the modules of SLED in the initial programming phases and as the program took its finished form. While not exhaustive the testing and evaluation performed by the programming team has resulted in a fully operational and effective editor.

Once the major operating bugs were identified and removed from the program, the task of specific debugging of each command and its interaction with the other commands of SLED was undertaken, including the testing of pathological errors where purposely erroneous and improper commands were input with the express purpose of causing the system to perform incorrectly or fail.

While we are satisfied that the program will function as required,
there is further room for testing of the system. The limited time available precluded the testing of large files and extensive directories and the actual production of useable and functional files. Besides the testing of large files, a period of time should be spent by disinterested parties (actual users) in using SLED to produce files and testing the system. From this evaluation, any remaining system bugs should be easily identified and corrected.
SLED - PASCAL
Programmed by: Robert M. Burnham
              Ronald J. Coulter
              Scott W. Smart

Naval Postgraduate School, Monterey, California

Specification: 20 February 1980

Written: 12 March 1980

Compiled: 14 March 1980

Source Computer: PDP 11-50

Object Computer: PDP 11-50

Language: PASCAL

Implementation: UC Berkeley PASCAL

File Location: UNIX PWH /work/cs500/smart/sled.p

Editor File Name: sled.p
Abstract: SLED is a very simple general purpose text editor implemented in PASCAL and designed to be relatively transportable to other PASCAL systems. It performs a minimum of the usual text editing and display features found in a typical text editor.
DOCUMENTATION FOR THE OPERATION OF
THE BODY OF THE MAIN PROGRAM CAN
BE FOUND AT LINE 2376

program sled(input, menu, version, output, directory);
label 10,100;

const
bufsize = 50;
linesize = 120;
nill = -1;
namesize = 8;
errmsg = 'INVALID COMMAND';
blank = ' ';
/* DATA ENTRY ERROR**:PLEASE REENTER DATA *****/

type
line = record
  nextline : -1..bufsize;
  linestring : packed array[1..linesize] of char;
end;
buffer = array[1..bufsize] of line;
datastring = packed array[1..linesize] of char;
header = record
  firstline: integer;
  ptr : -1..bufsize;
end;

var
letter, c, ch, ct : char;
errcnt, textsize, curline : integer;
menu, version, mfile, tempfile, directory, tempd : text;
buff : buffer;
free : -1..bufsize;
head : header;
filename : packed array [1..namesize] of char
fileopen : boolean;
cmderror : boolean;
PURPOSE: This procedure writes the contents of the line buffer to the currently open file. _mfile_ is a dummy file name. During execution, this file name is replaced by a "path" to the UNIX file name specified during a rewrite or reset operation. File _tempfile_ is a temporary file used during read and write operations to restore the user's UNIX file. The temp file exists during execution as UNIX file _temp.2_. If execution terminates normally, this file is removed from the system.

The procedure reads from the user's file (_mfile_) to the temp file until the point where the buffer is to be inserted is reached. At this point, the contents of the buffer are written to the temp file. After the buffer lines have been transferred, the remaining lines of the user file are read and written out to the temp file. The temp file now contains the complete file. The entire temp file is then read and written to the user's file.

VARIABLES: _temofile_: scratch file
_mfile_: dummy name for user's UNIX file
_fname_: actual UNIX file name
_point_: pointer to the buffer

procedure writebuf;
  type pointer = integer;
  var i : integer;
  ch : char;
  point : pointer;

begin
  { read all lines preceding the first line of the buffer from the user's file to the scratch file }
rewite(tempfile);
reset(mfile, file);
for i := 1 to (head.firstline - 1) do
begin
while not eoln(mfile) do
begin
read(mfile, ch);
write(tempfile, ch);
end;
end;
set the list pointer to the first element of the buffer
and read all lines in the list from the buffer to the scratch file

point := head.ptr;
while point <> nil do
begin
i := 1;
while (i < linesize) and (buff(point).linestring[i] <> ct) do
begin
write(tempfile, buff(point).linestring[i]);
i := i + 1
end;
write(tempfile);
point := buff(point).nextline;
end;
(reset the buffer status record)
head.ptr := nil;
head.firstline := 0;
free := 1;
for i := 1 to bufsize - 1 do
buff[i].nextline := i + 1;
buff[bufsize].nextline := nil;

(read past the lines of the user's file which were written to the
buffer previously)

for i := 1 to 40 do
begin
while not eof(mfile) do
readln(mfile);
end;
(read all remaining lines from the user's file to the scratch file)

while not eof(mfile) do
  begin
    while not eoln(mfile) do
      begin
        read(mfile, ch);
        write(tempfile, ch);
      end;
    if not eof(mfile)
      then begin
        readln(mfile);
        writeln(tempfile);
      end;
  end;

(write the entire scratch file into the user's file)

rewrite(mfile, name);
reset(tempfile);
while not eof(tempfile) do
  begin
    while not eoln(tempfile) do
      begin
        read(tempfile, ch);
        write(mfile, ch);
      end;
    if not eof(tempfile)
      then begin
        readln(tempfile);
        writeln(mfile);
      end;
  end;
end  ( writebuf )
PROCEDURE READBUF

PURPOSE: This procedure reads a block of 40 lines from a user file into the buffer. Mfile is a dummy file name which is replaced during execution by a "path" to a UNIX file. Strtline is the first line of the file to be placed in the buffer. This line and the next 39 (if they exist) are read from the user's file and linked into the list.

VARIABLES: mfile: dummy filename for user's text file
fname: actual UNIX file name
strtcite: first line to be placed in buffer
textsize: total number of lines in user's file

```
function readbuf(strtline : integer);
var ch : char;
i,k : integer;
numline : integer;

begin

(determine number of lines to be placed in buffer)

numline := 40;
if (textsize + 1 - strtline) < numline
then numline := textsize + 1 - strtline;
reset(mfile,fname);
for i := 1 to (strtline - 1) do
readln(mfile);

{ insert lines into buffer }

for i := 1 to numline do
begin
k := i;
while not eoln(mfile) do
begin
read(mfile,ch);
buf[i],linestring[k] := ch;
k := k + 1;
end;
```

buffer.linestring[k] := ct;
if i > 1 then buffer[i-1].nextline := ii;
readln(file);
end;
buffer[numline].nextline := nil;
free := numline + 1;
for k := numline + 1 to bufsize - 1 do
  buffer[k].nextline := k+1;
  buffer[bufsize].nextline := nil;

{update buffer status record}

head.ptr := 1;
head.firstline := startline;
end;  (readbuf)
PROCEDURE FETCHLINE

PURPOSE: This procedure takes the number values produced by the Lineout procedure and causes the appropriate number of lines to be printed on the CRT screen. It is called by those procedures which need to enter the users buffer and extract part of the text. It, in turn, calls the procedures Readbuf and Writebuf which are part of text-buffer storage system. If the users request for text exceeds the actual size of the text, this procedure will print all the text that is available.

VARIABLES: The following variables are used in Fetchline:

x, y: counting numbers for iterative routines
p: number pointer to the next line
fline: from-line value
tline: to-line value

Global variables include:
textsize: the number of lines in the users text
curline: current line value
head.firstline: first line of the users text
head.ptr: pointer to the first line
buff.pointer].nextline: number value of the next line in the buffer
buff.pointer].linestring[p]: character value of the line in the buffer

BEGIN

if fline > textsize then
  fline := textsize

if tline > textsize

procedure fetchline ( fline, tline : integer );

var
  x, y, p, pointer : integer;

begin

if fline > textsize then
  fline := textsize

if tline > textsize

then
  tline := textsize;
if tline < head.firstline
  then
    begin
      writebuf:
      readbuf ( tline );
    end;
pointer := head.ptr;

{ searches for the from-line value in the users text. If not in the buffer, then the rest of the file is searched. }

for x := head.firstline to ( tline - 1 ) do
  begin
    pointer := buff[pointer].nextline;
    if pointer = nil
      then
        begin
          writebuf :
          readbuf ( x );
          pointer := head.ptr
        end;
  end;

{ print out each line, character by character, from the buffer to the CRT for the from-line/to-line values. If the buffer is exceeded, then the next block of text is read to the buffer. }

for x := tline to tline do
  begin
    y := x;
    write ( ' ', y, 5, ' ' );
    p := 11
    while (buff[pointer].lineseq
  end;
end (p < linesize) do
  begin
    write( buff[pointer].linestring[p] );
    p := p + 1
    end;
  writeln;
  if x < tline then begin
    pointer := buff[pointer].nextline;
    if pointer = nil then
      begin
        writebuf;
        readbuf(y);
        pointer := head.ptr
      end;
    end;
  end;
curline := tline;
end (fetchline)
PROCEDURE LINEGET

PURPOSE: This procedure, called by some of the main procedures of SLFD, reads the line number of the user input instructions, constructs a "from-line/to-line" pair of variables and checks for errors in user input. The user can designate the current line with a carriage return or a change terminator command, insert a value for any other line desired or specify lines from one number to another. The values produced by this procedure are passed back to the calling command. This procedure is called by Displayline and Screenline.

VARIABLES: The following local variables are used:

```plaintext
PROCEDURE LINEGET (var line, toline: integer; var screencheck: boolean)
var
num, tnum, temp: integer;
check: boolean;
```

begin
begin
  if not eoln(input) then begin
    read(c);
    if c in ('0'..'9') then begin
      repeat
        if eoln(input) then
          check := true;
        temp := 10 * temp + ord(c) - ord('0');
        num := temp;
      until not (c in ['0'..'9']) or (check) or (c = ct) end;
      if (check) or (c = ct) then begin
        line := num;
        tolne := line;
        end;
      temp := 0;
      end;
    end;
  if not eoln(input) then begin
    read(c);
    if c in ('0'..'9') then begin
      repeat
        if eoln(input) then
          check := true;
        temp := 10 * temp + ord(c) - ord('0');
        num := temp;
      until not (c in ['0'..'9']) or (check) or (c = ct) end;
      if (check) or (c = ct) then begin
        line := num;
        tolne := line;
        end;
      temp := 0;
      end;
    end;
  if not (check) and (c = comma) then begin
    if c = comma then
      end;
then
read (c);  
until ( eoln(input) ) or ( c = ct )
end;

if line > toline
then
begin
  writeln(error);  
  writeln(' Data is entered " from line, toline" ');  
  line := curline;  
  errcnt := errcnt + 1;  
  toline := line;
end;
errcnt := 0;
if line = 0
then
  line := curline;
end;  ( lineget )
PROCEDURE DISPLAYLINE

PURPOSE: This procedure displays lines of text from the user's file. The user requests the current line, a specific line or a group of lines with a starting value and ending value. The input command is < L value(value) >. The number commands are read by calling Lineget and the actual lines are fetched and printed by Fetchline. This procedure is merely a vehicle for the interaction of these two procedures.

VARIABLES: The following variables are used:

linefrom: from-line value
lineto: to-line value
check: error detecting variable

procedure displayline;
var

linefrom, lineto: integer;
check: boolean;

begin
Lineget (linefrom, lineto, check);
if linefrom = 0 then linefrom := 1;
if textsize <> 0 then
  fetchline (linefrom, lineto);
else writeln('0 lines in file');
if eoln(input) then begin
  readin;
  writeln('E>');
  end;
end;  (displayline)
PROCEDURE SCREENLINE

PURPOSE: This procedure displays 20 lines of text to the user. It can be started at the current line or at any line designated by the user. If the request exceeds the text size, then the screen will terminate with the last line of the text. The input command for displayline is <S value>. If improper data values are input, the checker will print only the current line with the error diagnostic printed by the Lineget procedure.

The procedure calls the procedures Lineget and Fetchline.

VARIABLES: The following local variables are used:

| linefrom: from-line value |
| lineto: to-line value |
| checker: boolean value passed by Lineget which causes only the current line to print instead of 20 lines. Done on error only |

Global variable used:

curline: current line

procedure screenline ;

var

linefrom, lineto : integer ;
checker : boolean ;

begin

linenet ( linefrom, lineto, checker ) ;
if linefrom = 0 then
    linefrom := linefrom + 1 ;
if linefrom <= lineto then
    lineto := linefrom + 20 ;

if checker = false then

{ Error routine for invalid data in Lineget procedure }
begin
  linefrom := curline;
  lineto := linefrom
end;

if textsize <> 0 then
  fetchline ( linefrom, lineto )
else writeln('0 lines in file');
if eoin(input) then begin
  readln;
  write('E>');
  end;

end;  ( screenline )
PROCEDURE APPENDLINE

PURPOSE: This procedure inserts lines into the buffer. The parameter STRTLINE is the line which the inserted line(s) is to follow. If this line is not currently in the buffer, the contents of the buffer are written out and the buffer is refilled beginning with the line STRTLINE. The procedure then continues to insert lines until the end of input symbol is reached. If the buffer is filled during this operation the contents of the buffer are written out and the last line inserted becomes the first (only) line in the buffer.

VARIABLES:

point: pointer to the line in the buffer the inserted lines are to follow
enpoin: pointer to the line immediately following the lines inserted
bufline: counter to keep track of the line number which point is pointing to

procedure appendline (var strtlne: integer);

begin

{ test if inserted lines are to be placed before the first line of text. If so, pointer is nil. If the file is empty the first inserted line becomes the first line of text and the buffer status is updated }

if strtlne = 0
then begin
if head.firstline > 1
then begin

writebuf;

end

if strtlne > head.firstline
then begin

end

else begin

test bufline := 1;

end

if strtlne = 0
then begin

end

end
readbuf(1);
end;
if head.firstline = 0
then begin head.ptr := nil;
head.firstline := 1;
end;
epoint := head.ptr;
point := nil;
end

{find the start line in the buffer}
else begin
if head.firstline > strtline
then begin
writebuf;
readbuf(strtline);
point := head.ptr;
end
else begin
point := head.ptr;
bufline := head.firstline;
while bufline < strtline do
begin
point := buff(point).nextline
{if start line not in file, get start line}
if point = nil
then begin
writebuf;
readbuf(strtline);
point := head.ptr;
bufline := strtline;
end
else begin
bufline := bufline + 1;
end;
end;
end;

{after the start line is found, set epoint to the next line}
if buff(point).nextline <> nil then
epoint := buff(point).nextline
else epoint := nil;
end;
done := false;
while not done do
begin
  i := 1;
  if not eoln(input) then
    read(inach)
    else inpch := ' ';
  text[i] := inpch;
  if inpch = ' ' then begin
  end;
  if not eoln(input) then begin
    i := i + 1;
  end;
  text[i] := inpch;
end;

(end of input, link epoint line to last inserted line)
if (i = 1) or (inpch = ct) then begin
  if point <> nil then
    buff(point).nextline := epoint;
end;
done := true;
end;

(write text line into buffer)
if not done then begin
  textsize := textsize + 1;
  while not eoln(input) and (inpch <> ct) and(i < linesize) do
begin
  i := i + 1;
  read(inach);
  text[i] := inpch;
end;
if eoln(input) then begin

readln;
write('['); text[i++] := ct;
end;
(if no free lines available, write out contents of buffer)

if free = nil then begin
  if point <> nil then
    buff(point).nextline := epoint;
  writebuf;
  readbuf(startline);
  point := buff(head.ptr).nextline;
  epoint := buff(point).nextline;
end;
if point <> nil then buff(point).nextline := free
else head.ptr := free;
point := free;
startline := startline + 1;
free := buff(free).nextline;
buff(point).linestring := text;
(update current line)
curline := curline + 1;
end;
end; { appendline }
PROCEDURE REPLACELINE

PURPOSE: This procedure serves two purposes. First it deletes all lines between the parameters strtline and endline (inclusive). Next procedure calls procedure appendline allowing the user to add any lines in place of the deleted ones. If the lines to be deleted are not currently in the buffer, the buffer is written out and the first line to be deleted becomes the first line of the buffer.

VARIABLES:

point: points to the first line to be deleted
point: points to the line after the last line to be deleted
bufline: the line to which point is pointing

procedure replaceline(var strtline, endline : integer);

type pointer = z..bufsize;

var point, epoint : pointer;
bufline : integer;
temp : integer;

begin
(find startline in buffer. If not in buffer write contents of buffer to output file and input 40 lines beginning with startline.)

if head.firstline > strtline
then begin
writeline;
readbuf(strtline);
point := head.ptr;
end
else begin
point := head.ptr;
bufline := head.firstline;

while bufline < strtline - 1 do
begin
point := buff(point).nextline;
if bufline < textsize then
  if point = nil then
    begin
      writebuff;
      readbuf(strtline);
      point := head.ptr;
      bufline := strtline;
      end
    else begin
      bufline := bufline + 1;
      end;
  end;
end;

{ determine which lines are to be replace and link these lines to the free list. If no new lines are to be added, reconnect the lines in the buffer without the deleted lines. (i.e. connect strtline - 1 to endline). If new lines are to be added, replace the startline with the first new line and append any following lines to it, then reconnect the following lines beginning with endline. }

epoint := point;                {find the endline}
for bufline := strtline - 1 to (endline - 1) do
  begin
    epoint := buff[point].nextline;
    if bufline < textsize then
      if epoint = nil then
        then begin
          buff[point].nextline := nil;
          writebuff;
          readbuf(outline);
          point := 0;
          epoint := head.ptr;
        end;
    end;
  end;

{ update the textsize and buffer status record }
textsize := textsize - (endline - strtline) - 1;
  temp := buff[point].nextline;
  buff[point].nextline := buff[point].nextline;
  buff[point].nextline := free;
  free := temp;
  strtline := strtline - 1;
  appendline(strtline)
end;
{ replace line }
PROCEDURE STRINGDISP

PURPOSE: This procedure searches the text between the parameters startline and endline for any lines containing an occurrence of the string <pattern>. If the string is found, the line containing it is displayed by calling procedure fetchline. The procedure first computes the next table to implement the Knuth, Morris, Pratt string search algorithm. This table is then used in determining how far to move the pattern along the line of text in the event of a non-match.

VARIABLES:
next : next table
point: pointer to the textline currently being evaluated.
opatlength: the length of the pattern

procedure stringdisp(var startline,endline : integer; var pattern: natstring; var patlength:integer);

begin
(compute next table for string matching procedure )

i := 0;
next [i] := 0;
j := 1;
while j < patlength do
begin
done := false;

repeat
  if i > 0 then
  if pattern[i] <> pattern[j]
  then i := next[i]
  else done := true;
until (i <= 0) or done;
i := i + 1;
j := j + 1;
if pattern[i] = pattern[j]
then next[i] := next[j]
else next[i] := j;
end;

{ find strline in the buffer }

if head.firstline > strline
then begin
  writebuf;
  readbuf(strline);
  point := head.ptr;
end
else begin
  point := head.ptr;
  bufline := head.firstline;
  while bufline < strline do begin
    point := buff(point).nextline;
    (if line is not in buffer, write out buffer
    and read in current line)
    if point = nil
    then begin
      writebuf;
      readbuf(strline);
      point := head.ptr;
      bufline := strline;
    end
    else begin
      bufline := bufline + 1;
    end;
  end;
end
while bufline <= enaoline do
begin
  text := buff(point).linestring;
  if i
  | := i;
  j := j;
end;
(begin pattern matching)

while (i <= patlen) and (j <= linesize)
and (text[j] <> ct) do
begin
  if pattern[i] = text[j]
  then begin
    i := i + 1;
    j := j + 1;
  end
  (keep matching)

else begin
  if next[i] > 0
    then i := next[i]
  else begin
    i := j;
    j := j + 1;
  end;
end

else begin
  if next[i] > 0
    then i := next[i]
  else begin
    i := j;
    j := j + 1;
  end;
end;

(pattern matches)
if i > patlen then fetchline(bufline, bufline);
point := buf[point].nextline;
bufline := bufline + 1;
if bufline < endline then
  if point = nil
    then begin
      writebuf;
      readbuf(bufline);
      point := head.ptr;
    end;
end;

end { stringdiso }
PROCEDURE STRINGREPL

PURPOSE: This procedure searches the user’s text file between the parameters strtline and endline for any occurrences of the string <pattern>. If <pattern> is found in a line, it is replaced by the string <string> and the resulting line is displayed. The procedure uses the Knuth, Morris, Pratt algorithm to match the pattern in the line. The next table is first computed. This table is then used in computing how far to move the pattern along the text line in event of a non-match.

VARIABLES: patlenath: length of the string <pattern>
strlenath: length of the new string <string>
text: a line buffer for the current line
next: next table
point: pointer to the current line

procedure strinorepl (var strtline, endline: integer;
var pattern, string: patstring;
var patlenath, strlength: integer);

type pointer = -1..bufsize;

var
i,j,k,m,temp,temp2: integer;
bufflen: integer;
next: array[-1..linesize] of integer;
point: pointer;
text: packed array[-1..linesize] of char;
found: boolean;
done: boolean;

begin

compute next table

i := 0;
j := 0;
next[1] := 0;
while j < patlenath do begin
  done := false;
  repeat
    if j > 0 then
      \( ... \)
if pattern(i) <> pattern(j) then i := next(i)
    else done := true;
    until (i <= 0) or done;
    i := i + 1;
    if pattern(i) = pattern[j]
        then next[j] := next[i]
        else next[j] := i;
    end;

{ find start line in buffer }

if head.firstline > strtl ine
    then begin
        writebuf;
        readbuf(strt line);
        point := head.ptr;
    end
    else begin
        point := head.firstline;
        while bufl1 ne <= endflne do
        begin
            point := buff(point).nextline;
            if point = nil
                then begin
                    writebuf;
                    readbuf(strt line);
                    point := head.ptr;
                    bufline := strtl ine;
                end
            else begin
                bufline := bufline +1;
            end
            end;
    end;

{ try to match pattern in text line }

while bufline <= endline do
    begin
        found := false;
        text := buff(point).line string;
        i := 1;
        j := 1;
        temp := 1;
        temp2 := 1;
        while (i <= patlen) and (j <= linesize)
and (text[i] <> ct) do
begin
if pattern[i] = text[i]
then begin
   i := i + 1;
   j := j + 1;
   (keep matching)
end;
else begin
   (no match)
   if next[i] > 0
   then i := next[i]
   else begin
      i := i;
      j := j + 1;
   end;
end;
{pattern match found—replace pattern with string}

if i > patlength
then begin
   found := true;
   for k := temp2 to (j - (patlength + 1)) do
      begin
         huff[linepoint], linestring[temp] := text[k];
         temp := temp + 1;
         end;
   m := 1;
   for k := (j - patlength) to ((j - patlength) +
      (strlength - 1)) do
      begin
         if temp < linesize then
            huff[linepoint], linestring[temp] := string[m];
            temp := temp + 1;
         end;
         temp2 := j;
         i := 1;
         end;
   (continue to search for pattern)
end;
if found then begin
   m := temp2;
   if temp < linesize then
      for k := (temp to (temp + (j - temp2)) do
         begin
            if k < linesize then
               huff[linepoint], linestring[k] := text[m];
            m := m + 1;
         end;
end;
{
{display new line }
fetchline(huflne, buflne);
end;
point := buff\[point\].nextline;
buflne := buflne + 1;
if buflne < endline
then if point = nil
then begin
writebuf;
readbuf(buflne);
point := head.ptr;
end;
end; {strinareal}
PROCEDURE APPENDCOM

PURPOSE: This procedure computes the start line for the insert procedure (appendlne). If the line number is defaulted, the current line is used. If the line number exceeds the number of lines in the text, or is otherwise invalid, an appropriate error message is returned.

VARIABLES:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lineno</td>
<td>start line for text insertion</td>
</tr>
<tr>
<td>inv</td>
<td>true if invalid command</td>
</tr>
</tbody>
</table>

PROCEDURE appendlne:

```plaintext
var com: char;
lineno: integer;
inv: boolean;

begin

lineno := 0;
com := ';'
inv := false;
if eoln(inout) or (com = ct) then if textsize > 0 then lineno := curline;
while not eoln(input) and (com <> ct) do begin
    read(com);
    if not (com in ('0'..'9') + [ct]) then begin
        (compute line number)
        if not eoln(input) then repeat read(com) until eoln(input);
        writeln(errmsg);
        errcnt := errcnt + 1;
        inv := true;
    end
    else if com <> ct then
    begin
        lineno := (10 * lineno) + (ord(com) - ord('0'));
    end;
    if not inv then
    begin
        if (textsize < lineno) then begin
            writeln(textsize:' lines in file');
            if not eoln(input) then repeat read(com) until eoln(input);
        end
    end
end
```
else begin
  if eoln(input) then begin
    readln;
    write('1>');
    end;
    appendline(lineno);
    errcnt := 0;
  end;

  if eoln(input) then begin
    readln;
    write('E>');
  end;
  curline := lineno;
end;  (appendcom)
**PURPOSE:**
This procedure computes the start line and end line for the line delete procedure (removeline).
If the line number is defaulted, the current line is deleted. If the end line is not specified, the end line is deleted. If an invalid line number is input, and error message is returned. If the start line or end line is greater than the text size the last text line is deleted.

**VARIABLES:**
- linestart : start line for deletion
- lineend : end line for deletion
- inv : true if invalid line number is inputed

---

```pascal
procedure removeline;

var
  com : char;
  first : boolean;
  linestart, lineend : integer;
  inv : boolean;

begin
  inv := false;
  linestart := 0;
  lineend := 0;
  first := true;
  com := 't';
  while not eof(input) and (com <> ct) do
  begin
    com := read(input);
    if not (com in ['0'..'9'] + ['a'..'z'])
    then read(input);
    if not eof(input) then repeat read(com) until eoln(input);
    writeln('error!
    errorcount := errorcount + 1
    inv := true;
    errorcount := errorcount + 1
  end;
  if com <> ct
  then if first (compute start line)
    then if com = 't'
    then first := false
```

---


else linest := (10*linest) + (ord(com) - ord('0'))
else if com = ',',
then begin
if not eoln(input) then repeat
read(com) until eoln(input);
writeln(errmsg);
errcnt := errcnt + 1;
inv := true;
end
end  
( compute end line)
else linend := (10*linend) + (ord(com) - ord('0')));
end;
if not inv then begin
if linest = 0 then linest := curline;  
(start line becomes current line)
if first = true then linend := linest)
if (textsize < linest) or (textsize < linend)
or (linend < linest)
then begin
writeln(textsize:1,' lines in file');
end
else begin
if eoln(input) and (textsize > 0)
then begin
readln;
write('>')
end;
if textsize > 0 then
replace(line(linest,linend)
else writeln('0 lines in file');
end;
end;
if eoln(input)
then begin
readln;
write('>')
end;
curline := linest;
end;  
( readcom )
PURPOSE: This procedure computes the start line, end line, and pattern for the string search procedure (stringdiso).
If the line number is defaulted, all lines in the user's file are searched. If the pattern string is defaulted, the procedure prompts the user. If an invalid line number is entered, an error message is returned. The procedure also computes the length of the pattern string.

VARIABLES:
- linest: start line
- linend: end line
- string: pattern to be matched
- stsize: length of the pattern

procedure stringdiso:

begin
  com := char;
  linest := linend := stsize := integer;
  string := patstring;
  first := true;
  stk := boolean;

  while not enln(input) and (com <> ct) do
    begin
      linest := 0;
      stk := false;
      linend := stsize := 0;
      first := true;
      stn := false;
      while not enln(input) and (com <> ct) do
        begin
          if not stk
            then if com = ct
                then begin
                    stk := true;
                    linest := "01";
                end
                else if not (com in ['0'..'9'] + ['1','1'])
                  then begin
                      if not enln(input) then repeat read(com)
                  end
              end
          end
      end
  end
end
until eoln(input);
writeln(errmsg);
errcnt := errcnt + 1;
inv := true;
end
else if first (compute start line)
   then if com = '.' then first := false
       else linext := (10*linext) + (ord(com)-ord('0'))
else if com = ',' then
   begin
      if not eoln(input) then repeat read(com)
          until eoln(input);
      writeln(errmsg);
      errcnt := errcnt + 1;
      inv := true;
      end (compute end line)
else linend := (10*linend) +
               (ord(com)-ord('0'))
else if com <> ct then (compute pattern string and string length) begin
   stsize := stsize + 1;
   strinl[stsize] := com;
   end;
if not inv then begin
   if stsize = 0 then
      (if no string prompt user) begin
         writeln('Enter string?');
         com := ' ';
         if eoln(input) then readln;
         while not eoln(input) and (com <> ct) do
             readln(com);
         if com <> ct then begin
            stsize := stsize + 1;
            strinl[stsize] := com;
            end;
         end;
   end;
   (check for line defaults and set to current line. If line is
    greater than text size, set to last line of text)
   if linext = 0 then begin
      lin := lin - 1;
      lin := textsize;
      end;
else if first
then linend := linest;
if linend > textsize then linend := textsize;
if linest > textsize then linest := textsize;
if (linend < linest)
  then begin
    if not eoin(input) then repeat read(com)
      until eoin(input);
    writeln(errcnt);
    errcnt := errcnt + 1;
  end
  else if textsize > 0 then begin
    stringin(linest, linend, string, srsize);
    errcnt := 0;
  end
  else writeln('0 lines in file');
end;
if eoin(input) then begin
  readin;
  write('
');
end;
curline := linend;
end;          ( dispcom )
PURPOSE: This procedure computes the start line, end line, pattern string, and new string to be inserted for the string replace procedure (stringrep).
    If the line number is defaulted, the current line is used. If an end line is specified, the start line is used. If the new string or pattern string is defaulted, the user is prompted. If an invalid line number is entered, an error message is returned.

VARIABLES:
    linest: start line
    linend: end line
    oldstring: pattern string to be replaced
    newstring: new string to be inserted
    oldsize: pattern string length
    newsize: new string length

procedure stringrep;

var
    com: char;
    linest, linend, oldsize, newsize: integer;
    oldstring, newstring, first, last, start, end, file, line, line2: boolean;
    inv: boolean;

begin
    inv := false;
    linest := 0;
    linend := 0;
    com := ' ';
    oldsize := 0;
    newsize := 0;
    first := true;
    last := true;
    file := true;
    line := true;

    while not end (input) and (com <> et) do
    begin
        readln;
        repeat

    end;


}
if not oldsta then if com = ct then begin
  oldsta := true;
  com := ' ';
end else if not (com in ('0'..'9') + ['!', '']) then begin
  writeln(errmsg);
  errcnt := errcnt + 1;
  inv := true;
end else if first then begin
  if com = '!' then first := false
  else linest := (10*linest) + (ord(com) - ord('0'))
else if com = '.' then begin
  if not eoln(input) then repeat read(com) until eoln(input);
  writeln(errmsg);
  errcnt := errcnt + 1;
  inv := true;
end else linend := (10*linend) + (ord(com) - ord('0'))
else if not newsta then begin
  if com <> ct then begin
    oldsize := oldsize + 1;
    oldstringextrsize + 1 := com;
  end else begin
    newsta := true;
    com := ' ';
  end else if com <> ct then begin
    newsize := newsize + 1;
    newstringextrsize + 1 := com;
  end;
end;
end if not line then begin
  if oldsize = 0 then if input prompt user then begin
    writeln(errmsg)
    writeln('in string?
end;
com := ' ';
while not eoln(input) and (com <> ct) do
begin
read(com);
if com <> ct
then begin
oldsize := oldstsize + 1;
oldstring(oldstsize) := com;
end;
end;
if newsize = 0  // (if no new string prompt user)
then begin
if eoln(input) then readln;
write('newstring?');
com := ' ';
while not eoln(input) and (com <> ct) do
begin
read(com);
if com <> ct
then begin
newsize := newsize + 1;
newstring(newstsize) := com;
end;
end;
end;
end;

(check for line defaults and set default values. If line number
is larger than text size, set line number to the last line
of text.)

if linest = 0 then linest := curline;
if first
then linend := linest;
if linest > textsize then linest := textsize;
if linend > textsize then linend := textsize;
if (linend < linest)
then begin
if not eoln(input) then repeat read(com)
until eoln(input);
writeln(errmsg);
errcnt := errcnt + 1;
end
else if textsize > 0 then begin
stringrep(linest,linend,oldstring,
newstring,oldstsize,newstsize);
errcnt := 0;
end
else writeln('0 lines in file');
end:
Curline := linend;
if eoln(inout) then begin
   reading
   writeln('Ex');
   end;
end;  ( strocom )
BASELINE IMPLEMENTATIONS OF THE STANDARD LINE EDITOR (SLED)

OCT 80  L A COX, R J COULTER, C F TAYLOR

UNCLASSIFIED NPS52-80-012
PROCEDURE GETFILE

PURPOSE: This procedure inputs the user's file name. The SLED directory is then searched for that file name. If the file name is found, the file is opened and textsize is set to the number of lines in the file. If the file name is not found, a new file is created and the file name is entered in the directory. The file <directory> is an explicit UNIX file which contains the current SLED directory. The file <temp> is a scratch file which is created during execution as UNIX file tem.1. The contents of the directory are read to the scratch file during the directory search (less the file to be opened, if found). This scratch file is then read to the directory with the updated number of text lines for the currently opened file in procedure close.

VARIABLES:
- directory: SLED directory file. The format for the file is <text size> <file name>.
- temp: scratch file
- name: user's UNIX file name to be opened
- textsize: number of lines in the file

procedure getfile:

var
- ch: char;
- found: boolean;
- index, j, k, i: integer;

begin

ch := ' '; 

textsize := 0;

index := 0;

(output the user file name)

for i := 1 to textsize do

name[i] := ch;

i := 0;
while not eoln(input) and (ch <> ct) do
    begin
        read(ch) until (ch <> ' ') or eoln(input));
        if ch = ct
        then
            i := namesize + 1
        else begin
            i := i + 1;
            if i <= namesize then name[i] := ch
        end;
    end;

{search for the user file name in the directory}

if name <> ' ' then begin
    reset(directory);  
    found := false;
    i := 1;
    while not eof(directory) and not found do begin
        textsize := 0;
        (if the file is found, compute textsize)

        reset
        read(directory,ch));
        if ch in ['0'..'9'] then
            textsize := (10*textsize) + ord(ch)-ord('0'))
        until ch = ' ';
        i := 1;
        while i <= namesize do begin
            read(directory,ch));
            if name[i] <> ch
            then begin
                i := namesize + 1
            index := index + 1
            end
            else i := i + 1
        end;
        if i = namesize + 1 then found := true
        else if not eof(directory) then
            readln(directory));
        end;
        if i = namesize + 1
        then begin
            writeln(textsize :1, ' lines in file ', name);
            reset(directory);
        end;
rewritet(temd);  
for i := 1 to index do  
begin  
  write the contents of the directory, minus the  
  first file just opened to the scratch file)  
  read;  
  read(directory,ch);  
  writet(temd,ch);  
  until eof(directory);  
  read directory;  
  writeln(temd);  
  end;  
  repeat  
    read(directory,ch) until eof(directory);  
    read directory;  
    while not eof(directory) do  
      begin  
        while not eof(directory) do  
          begin  
            read(directory,ch);  
            writet(temd,ch);  
          end;  
          read(directory);  
          writeln(temd);  
          end;  
          k := 1;  
          if textsize > 0 then  
( read the first 40 lines of text into the buffer)  
          readbuf(k);  
else begin  
          writeln('Creating file ",fname,"');  
          textsize := 0;  
          writet(temd);  
          reset(directory);  
          while not eof(directory) do  
          begin  
            while not eof(directory) do  
              begin  
                read(directory,ch);  
                writet(temd,ch);  
              end;  
              read(directory);  
              writeln(temd);  
              end;  
          writet(file,fname);
[update the buffer status record and free list]
    head.ptr := nil;
    head.firstline := 0;
    free := 1;
    for i := 1 to hufsize := 1 do
        buff[i].nextline := j + 1;
        huff[hufsize].nextline := nil;
    end;
    iflenp := true;
    if true
    else begin
        writeln('incorrect file name');
        end;
    end;  (netfile)
PROCEDURE CLOSE

PURPOSE: This procedure closes a previously opened text file. The contents of the scratch file <temp> are read into the SLPD directory <directory>. The text size of the open file and the file name are then entered as the last line of the directory.

VARIABLES:
- directory : SLPD file directory
- temp : scratch file
- fname : currently open file name
- textsize : number of lines in currently open file
- init : character representation of integer textsize

procedure close
var ch : char;
dint, i, j : integer;
number : array [1..5] of char;

begin
  for i := 1 to 5 do
    number[i] := ' ';
  writeln(' - closing file ', fname, ' - ');
  j := 5;

  (convert integer textsize to ASCII representation)
  if textsize = 0 then number[1] := '0'
  while textsize <> 0 do
    begin
      dint := textsize mod 10;
      textsize := dint + ord('0');
      textsize := textsize div 10;
      ch := chr(dint);
      number[j] := ch;
      j := j - 1;
      end;

  (insert text size into scratch file)
  for i := 1 to 5 do
if number[] <> ' ' then
  write(temd, number[])
write(temd, ' ')

(insert file name into scratch file)
for i := 1 to namesize do
  write(temd, name[i])
  writein(temd))

(write scratch file to SLED directory)
rewrite(directory)
reset(temd)
while not eof(temd) do
  begin
    while not eoln(temd) do
      begin
        read(temd,ch)
        writein(directory, ch)
      end
    end
  end
writein(directory)
end.

writein(directory)
end;  (close)
PROCEDURE OPEN

PURPOSE: This procedure opens a user inputted text file.
If a file is currently open, it is closed and
the new file then opened. If the file name is
not specified, a user prompt is generated. The
file is then opened by procedure GETFILE.

VARIABLES:
inv : true if command is invalid

procedure open;
var com : char;
inv : boolean;
begin
(close a previously named file)
if fileopen
then begin
close;
filename := false;
end;
rewrite(textod);
com := "";
inv := false;
while not eqln(input) and (com <> ct) do
begin
read(com);
if com <> ct
then begin
if not eqln(input) then repeat read(com) until
"\n\n" in(input);
write(input); %
count := count + 1
inv := true;
end;
if not

(if no file name, current user)

then begin
  if eolp(input)
    then begin
      write('filename?');
      readln;
      netfile;
    end
  else netfile;
end;
if eolp(input)
then begin
  readln;
  write('t>');
end;
errcnt := 0;
end : (open)
PROCEDURE QUIT

PURPOSE: This procedure closes the currently open file if one exists by calling procedure close.

VARIABLES:
- inv : true if command is invalid

procedure quit;
var ch : char;
inv : boolean;
begin
  ch := ' ';  // Initial value for ch
  inv := false;
  while not eoln(input) and (ch <> ct) do
    begin
      read(ch);
      if ch <> ct
        then begin
            if not eoln(input) then repeat read(ch) until eoln(input);
            writeln(errmsg);
            errcnt := errcnt + 1;
            inv := true;
          end;
    end;
  if not inv and filopen then begin
    filopen := false;
    errcnt := 0;
  end;
  if eoln(input) then begin
    readln;
    inv := true;
  end;
end;
PURPOSE: This procedure provides the RLFD user with a description of the various commands available in SLED. It can be called by the user typing "M" or is automatically called if two invalid commands in a row are submitted. The procedure functions by utilizing a file "menu" which contains the SLED command summary. The file is reset, a character is read, then written etc. until end is reached. Then skip down to the next line of "menu" write the line just read and repeat the process until end is reached.

A copy of the command menu is included in the program documentation.

procedure writemenu;
begin
reset(menu);
while not end(menu) do begin
  while not eoln(menu) do begin
    read(menu,ch);
    write(ch);
  end;
  eoln(menu);
  writeln
end;
if not cmderror then begin
  if not eoln(input) then begin
    repeat read(ch) until (ch = ct) or eoln(input);
  end;
end;
else write("E");
end;
( writemenu )
**PROCEDURE WRITEVERS**

**PURPOSE:** This procedure provides the SLFD user with a description of the program version. In SLED it can be called by the user typing a "$v$". The procedure functions by utilizing a file "version" which contains the version documentation. The file is reset; a character is read then written etc. until EOF is reached. Then skip down to the next line of "version", write the line just read and repeat the process until EOF is reached.

A copy of the version format is included in the program documentation.

```pascal
procedure writevers;

begin
    reset(version);
    while not eof(version) do begin
        while not eofn(version) do begin
            readn(version,ch);
            write(ch);
        end;
        readln(version);
        writeln;
    end;
    if not eofn(input) then repeat
        readn(ch) until (ch = el) or eofn(input);
    if eofn(input) then begin
        readln;
        writeln('EOF');
        end;
end;  // writevers
```
PURPOSE: This procedure is used to change the value of the logical terminator from its optional value of $S$. This is done by inputting the new value of the logical terminator from the console and assigning it to the variable "ct". At the same time, the 50 line buffer is scanned and the logical terminators within each line of the buffer are scanned and changed to the new value.

VARIABLES:

Variables i and j are used as counters in repetitive statements. Subj is used as a temporary holder of the new logical terminator.

procedure changeterm;
var i:integer;
j:integer;
s:char;

begin
    writeln('ENTER THE CHARACTER DESTINED');
    writeln('AS A NEW LOGICAL MESSAGE TERMINATOR');
    writeln('MUST BE A PRINTABLE CHARACTER');
    writeln('LOGICAL TERMINATOR IS ',ct);
    writeln;
    readln;
    writeln('>');
    for i := 1 to bufsize do
    begin
        for j := 1 to linesize do
        begin
            subj := huff[i].lstring[j];
            if subj = ct then begin
                end;
            huff[i].lstring[j] := letter
        end;
    end;
end;
ends
ct := letter
writein("LOGICAL IFKMINATOR CHANGED TO ", ct);
readin;
write('E>');
ends (channeterm)
PROCEDURE ERRORROUTINE

PURPOSE: This procedure is called whenever there are two consecutive input mistakes made by the operator. The command menu will be printed on the CRT screen to inform the user of the proper commands for data display and input. The errorcount is reset to zero whenever this procedure is called (errorcnt is a global variable). When a proper command is entered by the operator, the errorcount is also set to zero. This parameter can be changed to display the command menu less frequently by a small change in the main program.

procedure errorroutine;
begin
    writeln;
    errorcnt := 0;
en: if (errorroutine)
PURPOSE: The main program is utilized to select the appropriate text editing procedure utilizing the inputs from the console. It also performs the initial editing of the inputs. The program will read the first 1 or 2 letters input to the console (depending on command input) and determine which procedure must be entered to execute the desired command. This is accomplished with a sequential scan of a series of if statements in which the first letter and if necessary the second letter of the command input to the console is checked against the authorized initial letter or letters of SL/D commands. If the initial editing is successfully completed in one of the if statements, the procedure associated with that if statement is entered. The remainder of the command is edited in the called procedure. If the sequential scan gets through the last if statement and no match is found with a valid command, the following is done:

1. Error message printed
2. Increment error counter
3. Check value of the error counter: if large enough enter procedure to write command menu
4. Check if more commands follow; if not, read a line
5. Return to statement 10 in the main program and read the next command.

In each if statement associated with commands which would have follow-on parameters after the initial letter(s), the status of the error count is checked to determine the success of editing in the procedure and if necessary the error message or menu is printed. After every return from a procedure, the program goes to statement 10 and reads the next command except for the null command which causes the program to terminate.
VARIABLES: The following global variables are used in the main program:

- `errorCnt`: count of user generated input errors
- `fileError`: boolean used to check for proper input commands
- `fileOpen`: boolean check to ensure that a file is opened before allowing the procedures to operate
- `letter`: character input of the user

```
begin

ct := 'a';
fileopen := false;
writeln;
writeln('FILEOPEN
');
writeln('FILEOPEN FOR COMMAND MENU');
writeln('FILEOPEN FOR SYSTEM VENSION');
writeln;
errorCnt := 0;
cederror := true;
writeln('c>1');
10: read(letter);

if (letter = 'c') then begin
endif else begin
writeln('no file open
');
readin;
write('F>1');
endif

if errorCnt > 1 then erroutlnl
10:

if (letter = 's') then begin
endif else begin
writeln('no file open
');
readin;
write('F>1');
endif

if errorCnt > 1 then erroutlnl
10:

if letter = 'd' then begin
endif
```

```
write_menu:
caller := true:

errcnt := 0:
goto 10

if letter = 'V' then begin writevers:
end:
errcnt := 0:
goto 10

if letter = 'D' then begin open:
end:
errcnt := 0:
goto 10

if letter = 'Q' then begin quiet; goto 100:
end;
if letter = 'A' then begin read(letter);

if (letter = 'L') then begin if fileopen then appendcode
else begin
writein("no file open");
reading
writeOE2.0I
end:
if errcnt > 1 then errroutine
end;

else begin
errcnt := errcnt + 1:
writein(errrnt);
if errcnt > 1 then errroutine
else writeOE2.0I;
readin:
goto 10:
end;
if letter = 'S' then begin read(letter);

if (letter = 'S') then begin if fileopen then disacc:
else begin
writein("no file open");
reading
writeOE2.0I
end:
if errcnt > 1 then errrout
end;

end.
else begin
  errcnt := errcnt + 1;
  writeln(errmsg);
  if errcnt > 1 then errroutine
    else write('E>');
      readln;
end;
goto 10

if letter = 'C' then
begin
  read(letter);
  if (letter = 'T') then begin change;
    errcnt := 0;
    goto 10
  end;
  else begin
    errcnt := errcnt + 1;
    writeln(errmsg);
    if errcnt > 1 then errroutine
      else write('E>');
        readln;
    end;
goto 10
end;

if letter = 'R' then
begin
  read(letter);
  if (letter = 'L') then begin if fileopen then replace
    else begin
      writeln('no file open');
      readln;
      write('E>');
    end;
    if errcnt > 1 then errroutine
      goto 10
  end;
  else begin
    writeln('no file open');
    readln;
    write('E>');
    end;
    if errcnt > 1 then errroutine
      goto 10
  end;

  else begin
    errcnt := errcnt + 1;
    writeln(errmsg);
    if errcnt > 1 then errroutine
      goto 10
  end;
else write('E'> ));

readln;
end;

errcnt := errcnt + 1;
writein(errmsg);
if errcnt > 1 then erroutine
  else write('F'> ));
    if letter < ' ' then readln;

note 10;

100: writeln(' ');
end.  ( main program )


(************************************************************ END SLED ************************************************************)


 Nicht S14/PI00 PASCAL USES IN THIS VERSION OF SLED
Obtained from the UC berkeley PASCAL translator ( pl -a option )

s 240 - Two argument forms of reset and rewrite are non-standard
s 297 - Two argument forms of reset and rewrite are non-standard
s 1920 - Two argument forms of reset and rewrite are non-standard
s 172 - Two argument forms of reset and rewrite are non-standard


<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Distribution List</th>
</tr>
</thead>
</table>
| 1.  | 2      | Defense Documentation Center  
|     |        | Cameron Station  
|     |        | Alexandria, Virginia 22314 |
| 2.  | 2      | Library, Code 0142  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93940 |
| 3.  | 1      | Office of Research Administration  
|     |        | Code 012A  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93940 |
| 4.  | 30     | Chairman, Code 52Bz  
|     |        | Computer Science Department  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93940 |
| 5.  | 10     | Lyle A. Cox, Jr., Code 52C1  
|     |        | Computer Science Department  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93940 |
| 6.  | 1      | I. Larry Avrunin  
|     |        | David Taylor Naval Ship Research and Development Center (Code 18)  
|     |        | Carderock Laboratory  
|     |        | Bethesda, MD 20084 |
| 7.  | 3      | R. P. Crabb, Code 9134  
|     |        | Naval Ocean Systems Center  
|     |        | San Diego, CA 92152 |
| 8.  | 1      | G. H. Gleisner  
|     |        | David Taylor Naval Ship Research and Development Center (Code 18)  
|     |        | Carderock Laboratory  
|     |        | Bethesda, MD 20084 |
| 9.  | 3      | Kathryn Heninger, Code 7503  
|     |        | Naval Research Lab  
|     |        | Washington D. C. 20375 |
| 10. | 3      | Ronald P. Kasik, Code 4451  
|     |        | Naval Underwater Systems Center  
|     |        | Newport, RI 02840 |
11. Commander, Code 503  
Naval Air Development Center  
Warminster, Pennsylvania 18974

12. Mark Underwood, Code P204  
NPROC  
San Diego, California 92152

DTRSSRD  
Bethesda, MD 20084

NSWC  
Dahlgren, Virginia 22448

15. John Zenor, Code 31302  
Naval Weapons Center  
China Lake, California 92555
ATE ME