POLISH, A Fortran Program to Edit Fortran Programs

by

John Dorrenbacher, David Paddock
David Wisneski, and Lloyd D. Fosdick

Department of Computer Science
University of Colorado
Boulder, Colorado 80309

Report #CU-CS-050-76(Revised) May, 1976

* This work supported in part by National Science Foundation Grant DCR 75-09972.

ABSTRACT

POLISH is a program which will read ANS Fortran programs and rewrite them in a stylized format designed to be easy for the reader and conservative in its use of space. Although some editing features, such as spacing conventions about tokens are fixed, many are subject to user control. POLISH itself is written in ANS Fortran.

Keywords: Fortran text editor.

1. INTRODUCTION

The text editor POLISH reformats a FORTRAN program to give it systematic spacing, indentation, labelling, and other stylistic features desirable for the reader. The edited program has a different physical appearance but it is identical to the original program with respect to its execution characteristics. Figures 1, 2, and 3 illustrate the kind of editing done by POLISH. Style is a matter of taste, but almost any reasonable stylistic conventions will produce a program listing more pleasing to the eye and easier for the human reader to follow. Other FORTRAN text editors are TIDY [1], FORDOC [2], FORTREDIT [2]. Particular features of POLISH which distinguish it from others are that it is written in ANS FORTRAN (cf. Appendix D), no manual insertion of editing cues in the unedited text is necessary but such cues may be used, line widths may be controlled, and spacing between tokens is controlled. Since early in 1974 POLISH has been used to prepare the FORTRAN listings which appear in the Algorithms Department of the Communications of the ACM and more recently in ACM Transactions on Mathematical Software; see for example [3].

POLISH is a subroutine subprogram. Parameters which control certain editing features are passed to POLISH through COMMON storage. When called into execution it will read the unedited FORTRAN text from a file and write the edited FORTRAN text onto another file; both files are specified by the user. Aside from certain exceptions noted in Section 2, it is assumed that the text to be edited is a syntactically correct ANS FORTRAN program. There are some circumstances in which it is desirable to inhibit editing of a text segment; most typically in

```
C EXAMPLE 1.
C THIS IS A NONSENSE PROGRAM ILLUSTRATING SOME FEATURES OF EDITING
C DONE BY POLISH.
C
      COMPLEX   Z
      EQUIVALENCE  (C,D),(E,F) , (G , H )
      DATA C,A/3.0, 2.0/
    3 A=1.0
    7 B=3.0*C+D * E / F
      CD(I , J )=AB(I,J)-EF(I+1)
      DO 10 I=1,50
      X=X+Y
      C =C +D
      DO10K=1,40
      E=E+1.0
      DO  10  L=1,100
   10 M=M+1
      IF(A.LT. C)GOTO3
      IF(A .LE.D)GO TO 3
      IF(A .EQ. C)GOTO 3
      IF(A .LT. B  .AND. C.LE.D.OR.X.EQ.Y.OR.A+B.NE.C*D.AND. 6.0.LE.A-B)
     +GOTO3
      X=X ** 5+A
      X=- 3.0
      X=+ 3.0
      IF(.NOT. C .OR..NOT.D)GOTO7
      Z=(3.0,2.0)+(2.0, 4.0)
      STOP
      END
$
```

Figure 1.a: Program before processing by POLISH.

```
C EXAMPLE 1.
C THIS IS A NONSENSE PROGRAM ILLUSTRATING SOME FEATURES OF EDITING
C DONE BY POLISH.
      COMPLEX Z
      EQUIVALENCE (C,D), (E,F), (G,H)
      DATA C, A /3.0,2.0/
   10 A = 1.0
   20 B = 3.0*C + D*E/F
      CD(I,J) = AB(I,J) - EF(I+1)
      DO 50 I=1,50
        X = X + Y
        C = C + D
        DO 40 K=1,40
          E = E + 1.0
          DO 30 L=1,100
            M = M + 1
   30     CONTINUE
   40   CONTINUE
   50 CONTINUE
      IF (A.LT.C) GO TO 10
      IF (A.LE.D) GO TO 10
      IF (A.EQ.C) GO TO 10
      IF (A.LT.B .AND. C.LE.D .OR. X.EQ.Y .OR. A+B.NE.C*D .AND.
     * 6.0.LE.A-B) GO TO 10
      X = X**5 + A
      X = -3.0
      X = +3.0
      IF (.NOT.C .OR. .NOT.D) GO TO 20
      Z = (3.0,2.0) + (2.0,4.0)
      STOP
      END
```

```
MAN    10
MAN    20
MAN    30
MAN    40
MAN    50
MAN    60
MAN    70
MAN    80
MAN    90
MAN   100
MAN   110
MAN   120
MAN   130
MAN   140
MAN   150
MAN   160
MAN   170
MAN   180
MAN   190
MAN   200
MAN   210
MAN   220
MAN   230
MAN   240
MAN   250
MAN   260
MAN   270
MAN   280
MAN   290
MAN   300
MAN   310
```

Figure 1.b: Program after processing by POLISH.

```
C EXAMPLE 2.
C THIS IS ANOTHER NONSENSE PROGRAM ILLUSTRATING SOME FEATURES OF
C EDITING DONE BY POLISH.
C
      X=1.0
C     OBSERVE HOW THIS BLOCK OF COMMENTS
C     IS LEFT ADJUSTED, PRESERVING THE
C     RELATIVE INDENTATION.
      Y=2.0
C     THIS BLOCK HAS A COMMENT LINE WHICH IS TOO LONG (CF. KEYS(1)) SO BREAK
C    AND APPEND PROCEDURES ARE APPLIED. THIS MAY NOT ALWAYS HAPPEN AND
C    DEPENDS ON THE KEYS PARAMETER VALUES.
      Z=3.0
C     THIS BLOCK ALSO HAS A COMMENT LINE WHICH IS TOO LONG, HOWEVER BREAK
C     AND APPEND PROCEDURES ARE NOT APPLIED IN THIS CASE BECAUSE THIS
C     COMMENT BLOCK CONSISTS OF MORE THAN THREE LINES. NOTE THAT THE
C     DEFAULT VALUE FOR KEYS(10) IS 3.
      Z=1.0
      WRITE(6,10)
   10 FORMAT(*NOTE REPOSITIONING OF THIS STATEMENT*)
      WRITE(6,20)
   20 FORMAT(*NOTE STRING DELIMITER REPLACEMENT*)
      WRITE(6,30)X,Y,Z
   30 FORMAT(3H X=,E20.10,3H Y=,E20.10,3H Z=,E20.10/*OBSERVE THE SPLIT
     +WHICH IS DONE HERE*)
      STOP
      END
$
```

Figure 2.a: Program before processing by POLISH.

```
C EXAMPLE 2.                                                                    MAN    1
C THIS IS ANOTHER NONSENSE PROGRAM ILLUSTRATING SOME FEATURES OF                MAN    2
C EDITING DONE BY POLISH.                                                       MAN    3
      X = 1.0                                                                   MAN    4
C  OBSERVE HOW THIS BLOCK OF COMMENTS                                          MAN    5
C IS LEFT ADJUSTED, PRESERVING THE                                              MAN    6
C RELATIVE INDENTATION.                                                         MAN    7
      Y = 2.0                                                                   MAN    8
C  THIS BLOCK HAS A COMMENT LINE WHICH IS TOO LONG (CF.                         MAN    9
C KEYS(1)), BREAK AND APPEND PROCEDURES ARE APPLIED. THIS MAY                   MAN   10
C NOT ALWAYS HAPPEN AND DEPENDS ON THE KEYS PARAMETER VALUES.                   MAN   11
      Z = 3.0                                                                   MAN   12
C THIS BLOCK ALSO HAS A COMMENT LINE WHICH IS TOO LONG, HOWEVER BREAK           MAN   13
C AND APPEND PROCEDURES ARE NOT APPLIED IN THIS CASE BECAUSE THIS               MAN   14
C COMMENT BLOCK CONSISTS OF MORE THAN THREE LINES. NOTE THAT THE                MAN   15
C DEFAULT VALUE FOR KEYS(10) IS 3.                                              MAN   16
      Z = 1.0                                                                   MAN   17
      WRITE (6,99999)                                                           MAN   18
      WRITE (6,99998)                                                           MAN   19
      WRITE (6,99997) X, Y, Z                                                   MAN   20
      STOP                                                                      MAN   21
99999 FORMAT (36HNOTE REPOSITIONING OF THIS STATEMENT)                          MAN   22
99998 FORMAT (33HNOTE STRING DELIMITER REPLACEMENT)                             MAN   23
99997 FORMAT (3H X=, E20.10, 3H Y=, E20.10, 3H Z=,                              MAN   24
     * E20.10/37HOBSERVE THE SPLIT  WHICH IS DONE HERE)                         MAN   25
      END                                                                       MAN   26
```

Figure 2.b: Program after processing by POLISH.

```
C EXAMPLE 3.
C THIS EXAMPLE ILLUSTRATES ALPHABETIZATION OF SUBPROGRAMS.
      X=2.0
      READ(5,10)X
   10 FORMAT(F10.0)
      CALL ASUB(X)
      STOP
      END
      SUBROUTINE BSUB(U,V)
      V=U+1.
      X=Y
      RETURN
      END
      SUBROUTINE ASUB(Y)
      Y=4.0
      CALL BSUB(Y,R)
      RETURN
      END
$
```

Figure 3.a: Program before processing by POLISH.

```
C EXAMPLE 3.                                                    MAN   10
C THIS EXAMPLE ILLUSTRATES ALPHABETIZATION OF SUBPROGRAMS.      MAN   20
      X = 2.0                                                   MAN   30
      READ (5,99999) X                                          MAN   40
      CALL ASUB(X)                                              MAN   50
      STOP                                                      MAN   60
99999 FORMAT (F10.0)                                            MAN   70
      END                                                       MAN   80
      SUBROUTINE ASUB(Y)                                        ASU   10
      Y = 4.0                                                   ASU   20
      CALL BSUB(Y, R)                                           ASU   30
      RETURN                                                    ASU   40
      END                                                       ASU   50
      SUBROUTINE BSUB(U, V)                                     BSU   10
      V = U + 1.                                                BSU   20
      X = Y                                                     BSU   30
      RETURN                                                    BSU   40
      END                                                       BSU   50
```

Figure 3.b: Program after processing by POLISH.

COMMENT blocks that have already been formatted in a special way. The editing will be suspended by the appearance of a card with an asterisk in column 1 and will resume with the second appearance of such a card. Also, parameter specifications and delimiters can be used to suspend all editing of COMMENT statements or limit it to indicated areas.

POLISH is a descendant of a program called STYLE written by Dorothy Lang Wedel [4]. While POLISH contains many of the design goals of STYLE, it is an entirely new program. POLISH uses Sale's algorithm to identify Fortran statement types [5]. Most of the production work on POLISH was done by a group of twelve undergraduate students at the University of Colorado as a class project, supervised by one of the authors [LDF]. The coauthors of this report were the principal contributors in this project.

The general operating requirements and run time data for POLISH on a CDC 6400 operating under Kronos 2.1 without subprogram alphabetization are*:

(1) Field length to run- - - - - - - 57000 (octal)

(2) Files on secondary storage - - - 8

(3) Processing time- - - - - - - - - 30.3 statements/CPU sec

(4) Job costs- - - - - - - - - - - - $1.50 (100 statements)
$2.50 (200 statements)
$3.20 (300 statements)

* This data depends on the environment. It is correct for the systems environment at the Computing Center of the University of Colorado, September 14, 1976.

2. EDITING FEATURES

Fixed editing features. There are three editing features, not subject to user control:

(1) Spacing conventions between tokens;

(2) Renumbering of statement labels and removal of unreferenced labels;

(3) Insertion of CONTINUE statements to force termination of every DO-block on a unique CONTINUE statement.

The goal of the spacing conventions is to produce a text pattern which is easy for the reader of FORTRAN text to follow, and at the same time is economical in its use of blanks. An important principal in this connection is that the token sequence <identifier> <operator> <identifier> is grouped closer together for operations executed first (in the execution hierarchy) than for others, except inside parentheses. A list of spacing conventions used in POLISH appears below:

(1) No space before or after operators within parentheses except before and after logical operators .AND. and .OR.;

(2) No space before or after the operators *, **, /, .LT., .LE., .EQ., .NE., .GE., .GT.;

(3) Single space before and after the operators +, -, =, .AND., .OR. except in DO statement and within parentheses (cf.1);

(4) No space after the unary operators +, -, .NOT.;

(5) One space to the right of the comma, except in complex constants, subscripts, within solidus delimiters of a DATA statement, and within parenthesis delimiters of an EQUIVALENCE statement;

(6) No space to the right of a left parenthesis or a solidus as a left delimiter;

(7) No space to the left of a right parenthesis or a solidus as a right delimiter;

(8) No space between nested parentheses;

(9) No space within any subscript;

(10) Normal English spacing is used elsewhere, except no space to the right of a solidus used as a field separator in a FORMAT statement.

Certain commonly used, but non-standard, FORTRAN constructions are accepted by POLISH. These are:

(a) Hollerith strings in FORMAT statements delimited by a special symbol (e.g. *SAMPLE* instead of 6HSAMPLE);

(b) FORTRAN II READ, PRINT, and PUNCH statements (e.g. READ 99, X instead of READ (5,99) X);

(c) DATA statement for array elements without explicit naming of the elements (e.g. DATA A/5.0, 3.0, 1.0/ instead of DATA A(1), A(2), A(3) / 5.0, 3.0, 1.0 /)

POLISH will replace the Hollerith string with delimiters in a FORMAT statement by the standard form; thus *SAMPLE* would be replaced by 6HSAMPLE.

Labels are renumbered so that they increase by a fixed amount in order of their appearance on statements. The fixed increment can be set by the user (cf. KEYS(11) description). While unreferenced labels do not violate the ANSI standard, they are a source of confusion and may not be accepted by a compiler. POLISH removes unreferenced labels.

User controlled editing features. There are a number of editing features under control of the user. Each of these features is designated by a parameter value passed to POLISH in a thirteen element array in labelled COMMON; viz.

COMMON/PARMS/KEYS(13)

A description of each element of the KEYS array follows. Within POLISH each element of KEYS is initialized by a DATA statement; these initial values thus serve as default values. It should be noted that once these values are changed by a user in the course of execution of the program, the "default" value is permanently overwritten for that execution run.

KEYS	DESCRIPTION	DEFAULT	RANGE[+]
1.	Rightmost card column in which FORTRAN code can appear; subsequent code is put on continuation cards with an * in card column 6.	64	[32,72]
2.	Left margin for comment statements.	3	[3,KEYS(1)-29]
3.	Card column for right justification of statement labels.	5	[1,5]
4.	Increment for statement labels.	10	>0
5.	Number of columns indented on continuation cards.	1	≥0
6.	Increment for sequencing in card columns 73-80. If 0 then no sequencing. Sequencing consists of the first three letters of the name of the subprogram and a sequence number. POLISH assigns the name MANPRG to a main program and BLKDAT to all BLOCK DATA subprograms.	10	≥0
7.	Number of columns indented in DO loops. POLISH indents all code within DO loops according to this specification. Note: Indentation of DO loops or continuation cards cannot exceed 20 spaces to the left of the right margin. If this occurs, then indentation is set to KEYS(1)-20.	2	≥0
8.	Blank comment cards removed. 0--- they are removed, anything else they are not.	0	N/A
9.	Authorization to break and append comments. If 0--- as permitted by KEYS(10) and + delimiters, anything else --- break and append comments to conform to KEYS(1) and KEYS(2).	0	N/A
10.	Maximum size of non-delimited comment block in which break and append procedures may be used when KEYS(9)=0.	3	≥0
11.	Optional Hollerith delimiting character Effect is to replace say, *SAMPLE* by 6HSAMPLE in FORMAT statements only.	*	N/A
12.	Edit comments. 0--- yes, anything else---no. If no then KEYS(2), KEYS(8), KEYS(9) and KEYS(10) are ignored, except to check if KEYS(2) is in range. If it is not in range an error message is printed and execution is terminated.	0	N/A
13.	Alphabetize subprograms. 0--- yes, anything else--- no.	0	N/A

[+][a,b] means all integers in this interval including a and b.

3. INPUT AND OUTPUT

Seven files are used by POLISH for input, output, and intermediate storage. Unit numbers for these files may be passed to POLISH through labeled COMMON, otherwise default values are used. The specification of the unit numbers in labeled COMMON, the file function, and the default value is given below.

COMMON/INOUT/K1, K2, K3, K4, K5, K6, K7, K8

File	Function	Default Value*
K1	Input file. Contains program to be edited. End of file flag is card with $ in column 1. This file must be positioned before POLISH is called, POLISH does not rewind this file. File read in 80A1 format.	1
K2	Scratch file for COMMENT statements. File read and written in 80A1 format.	2
K3	Scratch file. File read and written in unformatted binary.	3
K4	Scratch file for FORMAT statements. File read and written in unformatted binary.	4
K5	Not used in present implementation.	5
K6	Scratch file for subprogram alphabetization. File read and written in 80A1 format.	6
K7	Output file. The edited program is written on this file. POLISH does not rewind this file. No carriage control characters are written on this file. File is written in 80A1 format.	7
K8	Error message file. Error messages generated by POLISH are written on this file. POLISH does not rewind this file. Carriage control characters are written on this file. File is written in A1 format.	8

* Default values are set by BLOCK DATA

FORMAT statements in the program text may be moved to the end of the subprogram block, immediately before the END statement, or left in place. Movement to the end of the subprogram block is achieved by assigning different numbers to files K3 and K4; explanation of file specification appears in the next section. FORMAT statements are left in place if the same number is assigned to files K3 and K4.

FORMAT statements are renumbered starting at 10**KEYS(3)-1, and decreasing by 1. Thus for KEYS(3)=5 the FORMAT statements are numbered 99999, 99998, 99997, ... Unreferenced FORMAT statements are given a blank label and an error message is printed.

A card with an asterisk in column 1 will suspend all editing of the cards which follow except for the generation of sequence numbers; thus columns [1, 72] will appear in the output identically as input. The card with the asterisk in column 1 is not reproduced on the output. Editing is thus suspended until the next appearance of a card with an asterisk in column 1. Suspension of editing can be repeated. This feature is useful for preserving the layout of COMMENT statements.

If editing is suspended as just described on a segment of code containing statement labels, unreferenced labels or duplicate labels may result since labels are ignored while editing is suspended.

4. COMMENT STATEMENT EDITING

COMMENT statements are edited in blocks of 1 to 120 statements. A COMMENT block begins with the first COMMENT after a non-comment statement and is terminated by a non-comment statement or by one of the editing delimiter cards. The 120th COMMENT will terminate a block and the 121st COMMENT start another block. A delimited COMMENT table is processed as a block regardless of length.

COMMENT statement editing is controlled by six KEYS parameters (cf. Section 2, KEYS description) and three sets of delimiters. In decreasing order of precedence the editing controls are:

Editing Control	Use	Result of Editing Control Use
* in card column one	Do not edit delimiter	Columns (1,72) of COMMENTS and other statements appearing between * delimiter cards are copied directly with no editing. Sequence numbers are generated and added in columns (73,80) in accordance with KEYS(6).
KEYS(12)	Indicates COMMENT editing is permitted/not permitted	For a non-zero value, each COMMENT block is treated as if it were delimited by do not edit delimiters. For a zero value COMMENT blocks are edited as specified by other COMMENT editing controls.
KEYS(1)	Indicates right margin for edited COMMENT statements	Point at which COMMENT is to be broken if required and permitted. Error message is issued if edited COMMENT exceeds KEYS(1) value.
KEYS(8)	Indicates blank COMMENTS are to be retained/deleted	For a zero value, blank COMMENTS are deleted. For a non-zero value blank COMMENTS are retained.
++ in card columns one and two	Table delimiter	COMMENTS appearing between ++ delimiter cards are processed as a table in accordance with the data card immediately following the initial ++ delimiter card (cf. Appendix C, COMMENT Table Processing).
KEYS(2)	Indicates left margin for edited COMMENT statements	Indented COMMENTS are indented from this value.

KEYS(9)	Indicates specially formatted COMMENT statements may be/ are not present	For a non-zero value specially formatted COMMENTS are not present, COMMENTS may be broken and appended to conform to KEYS(1) and KEYS(2). For a zero value, specially formatted COMMENTS may be present. COMMENT blocks may be broken and appended only if permitted by + delimiters or KEYS(10).
+ in card column one	COMMENT edit delimiter	COMMENTS appearing between + delimiter cards may be broken and appended to conform to KEYS(1) and KEYS(2). FORTRAN code, delimited COMMENT tables and do not edit delimited statements may be nested, in the sense of DO loop nesting, between the + delimiter cards.
KEYS(10)	Indicates maximum size of non-delimited COMMENT block in which break and append procedures may be used when KEYS(9)=0	COMMENT blocks of length $\leq$ KEYS(10) may be broken and appended to conform to KEYS(1) and KEYS(2). Primary purpose of this is to permit editing of small COMMENT blocks, which presumably do not have a tabular structure which needs to be preserved.

For non-delimited and + delimited COMMENT blocks, editing is divided into minimum, advanced and blank COMMENT editing phases. Minimum editing consists of shifting COMMENT blocks en bloc to conform to KEYS(2). If the COMMENT block does not then conform to KEYS(1), an attempt is made to achieve conformance by reducing relative indentations. If the attempt is successful, indentations are changed and there is no need for advanced editing. If the attempt is not successful the COMMENT lines are left in the shifted position as required by KEYS(2), indentations are not changed, and if advanced editing is not permitted an error message is issued; in the latter instance, the right end of COMMENT statements may be lost. Advanced editing consists of breaking and appending COMMENT statements so that the block will conform to KEYS(1) (cf. Appendix B). Blank COMMENTS are edited in accordance with KEYS(8).

5. CALLING PROCEDURE

If default values for KEYS and file numbers are used, then the main program is simply

```
CALL POLISH
STOP
END
```

If FORMAT statements are to be left in place but otherwise default options are used, then the main program is

```
COMMON/INOUT/K1, K2, K3, K4, K5, K6, K7, K8
K3=K4
CALL POLISH
STOP
END
```

If no editing of COMMENT statements is desired, FORMAT statements are to be left in place and subprograms are not to be alphabetized but otherwise default options are used, then the main program is

```
COMMON/PARMS/KEYS(13)
COMMON/INOUT/K1, K2, K3, K4, K5, K6, K7, K8
K3=K4
KEYS(12)=1
KEYS(13)=1
CALL POLISH
STOP
END
```

POLISH can be used to edit more than one program, with different editing features selected. For example:

```
COMMON/PARMS/KEYS(13)
COMMON/INOUT/K1, K2, K3, K4, K5, K6, K7, K8
CALL POLISH
K3=K4
CALL POLISH
KEYS(12)=1
CALL POLISH
STOP
END
```

This main program will cause the editing of three programs on the input file: The first program will be edited with all default options; the second program will have FORMAT statements left in place; the third program will have FORMAT statements left in place and COMMENT statements will not be edited.

Input file. The input file must contain a card with a $ in column 1 at the end of the program to be edited by a call to POLISH. In the third example above three programs are processed by POLISH and each would have to be followed by a card with a $ in column 1.

6. ERROR MESSAGES

Two types of error messages are generated by POLISH.

TYPE 1 FORMAT

ERROR NN WAS DETECTED IN XXX.

ERRSUB WAS CALLED BY YYY.

Where:

NN is the error number.

XXX is user Routine name

YYY is name of POLISH routine that detected the error.

TYPE 2 FORMAT

****** ERROR TYPE NN HAS BEEN DETECTED IN THE ABOVE STATEMENT.

THIS STATEMENT IS IN ROUTINE XXX STATEMENT NUMBER III.

Where:

NN is the error number.

XXX is the name of the users routine.

III is the sequence number of the bad statement.

ERROR NUMBERS FOR TYPE 1 ERRORS.

Routine in which detected	Error number	Meaning	Comments
LABEL	-1	too many statement labels in one subroutine	length of DEF array must be increased with MAXDEF changed to reflect this change. Length is now 100.
	-2	too many reference labels in one subroutine	length of REF array must be increased with MAXREF changed to reflect this change. Length is now 200.
	-3	DO statements nested too deep	length of DOSTCK array must be increased with MAXSTK changes to reflect this increase. Length is now 10.
	-4	too many DO statements in one subprogram	length of DOARRY array must be increased with MAXRRY changed to reflect this increase. Length is now 50.
ENDLOP	-1	Same as LABEL -2	
	-2	Same as LABEL -1	
BCKSCH	±1	Program error	Should never occur.
DEFPRO	-1	Same as LABEL -1	
	2	renumbering of FORMAT statements overlaps with renumbering of other statement. Non-fatal.	Adjust KEYS(3) and KEYS(4) and rerun.
DSORT	-1	Program error.	Should never occur.
	-2	Same as LABEL -2.	
KSETA	-1	Program error.	Should never occur.
KSETB	-1	Program error.	Should never occur.
KDECNA	0,-1	Program error.	Should never occur.
KADVBA	-1	Program error.	Should never occur.
KADVBB	-1	Program error, or B array overflow.	Should never occur. See note below.
KDECBA	-1	Program error.	Should never occur.
KADVNA	-1	Program error.	Should never occur.

Some of the program errors which normally should never occur, may occur if the input to POLISH is not recognizable by POLISH.

NOTE: POLISH can input up to 19 continuation cards with no errors. This is equivalent to 1326 characters (72 characters from the first card and 66 from each continuation card). During processing POLISH inserts and deletes blanks and renumbers statement labels. If the number of characters ever increases over 1326 then the B array overflows. To correct this the statements should be broken up into two or more statements, or the A and B arrays must be increased with MAX (located in labeled common DEMMAX) also changed to reflect this increase.

All above errors are fatal except as noted.

ERROR NUMBERS FOR TYPE 2 ERRORS.

All errors non-fatal except 25.

Error Number	Meaning
1	non-ANS use of a letter.
2	non-ANS use of a number.
3	This is a program error and should never occur.
4	non-ANS use of a left parenthesis.
5	non-ANS use of a right parenthesis.
6	non-ANS use of a comma.
7	non-ANS use of an equals sign.
8	non-ANS use of a plus or minus sign or non-ANS use of a character not included in errors 1-7.
9	non-ANS use of an asterisk or solidus.
10	non-ANS use of a period.
11	This is a program error and should never occur.
12	non-ANS use of a character not included in errors 1-11.
13	This is a program error and should never occur.
14-16	Not used.
17	non-ANS useage may have induced error into statement containing Hollerith string.
18-19	Program error. Possible non-ANS statement.
20	non-ANS statement, or second portion of LOGICAL IF statement is non-ANS, or program error.
21	Second delimiter missing in FORMAT Hollerith string.

22-23	Program error. Possible non-ANS statement.
24	Too many continuation cards on output. Adjust KEYS(1).
25	Error in values of KEYS. FATAL.
26	Non-digit entry in statement label position.
27	Program error. Possible non-ANS statement.
28	More than 19 continuation cards on input.
29	Hollerith string too long to fit on one card image. Adjust KEYS(1) to make card image bigger.
30	Possible improper nesting of DO statements.
31	Missing statement label. Slashes are inserted wherever this label is referenced.
32	Statement label exceeds 5 characters after renumbering, or statement label exceeds number permitted by KEYS(3). Asterisks are inserted.
33	A FORMAT statement is unreferenced.
34	Same as 31 but applies only to FORMAT statements.
35	Same as 32 but applies only to FORMAT statements.
36	Statement is unrecognizable by POLISH.
37	Not used.
38-39	Program error. Possible non-ANS statement.
40	Sequence number greater than 9999. Restart at KEYS(6).
41	Zero-length Hollerith string.
42	This comment block contains at least one COMMENT with length exceeding KEYS(1).
43	Unable to break COMMENT within maximum authorized text length.
44	Unable to break last column of COMMENT table where indicated.
45	Length of table line exceeds KEYS(1).
46	Length of table line exceeds KEYS(1). Line not broken because corrected column number for continuation exceeds column 72.
47	Table processing terminated by a non-comment statement not located between do not process delimiters. If terminating table delimiter now encountered errors including loss of next card image will be introduced possibly without warning.
48	Label not referenced – blanks are inserted.

WARNING:

When an error is detected in one statement, the user is cautioned that POLISH has a tendency to spread this error to other statements without giving error messages. Therefore, when an error is detected, the user should fix the offending statement and re-run the whole subprogram through POLISH. An example:

```
      FUNCTION WRONG(A)
C  THE FOLLOWING SHOULD BE GO TO
      GTO 10
      RETURN
10    WRONG = 2.0*A
      RETURN
      END
$
```

POLISH will detect an error in the GO TO statement and hence not process it. Therefore, statement label 10 appears to be unreferenced, and POLISH will delete it.

A NOTE ABOUT HOLLERITH STRINGS:

A Hollerith string will not be detected unless the string follows a , (/ * All ANS FORTRAN input will have this condition satisfied. Generally if an error is detected in a statement containing a Hollerith string, the output will not be correct in that the string length is missing and the letter H is shifted to the left.

7. MACHINE INDEPENDENCE

SUBROUTINE TYPE is written to be completely machine independent and is therefore very inefficient requiring an average of thirteen comparisons to classify a character as a letter, thirty-one comparisons to classify as a digit, and forty-one to classify as a blank or special character. A listing of SUBROUTINE TYPE is provided in Appendix A to assist users in preparing an efficient machine dependent version of this intensively used routine which classifies each character in a non-comment statement.

Alphanumeric character storage is one character per word for data and program constants with the following exceptions:

a. End of statement mark, 3HEOS

b. Statement label flag, 2HFG

c. Hollerith string flag, 2HH*

d. Error messages contained in DATA statements in the various routines and printed by SUBROUTINE ERRSUB have 4Hxxxx Hollerith specifications.

The entire program is input/output bound with four mass storage accesses per statement and two additional mass storage accesses for each COMMENT or FORMAT statement if they are edited in any way. Mass storage accesses are nearly continuous during subprogram alphabetization. Installations where mass storage access are relatively expensive may desire to use machine dependent input/output buffering routines.

REFERENCES:

[1] TIDY, University of Colorado Computer Center program Q-TIDY-237. (Adapted from program by Harry M. Murphy, Air Force Weapons Laboratory, Kirtland Air Force Base, New Mexico 1966).

[2] FORDOC, and FORTREDIT are propietary software products. See ICP Quarterly, Vol. I, October, 1973, pp. 201 and 174.

[3] Algorithm 476, Comm. ACM 17 (April 1974), 220-223.

[4] Lang, Dorothy E. STYLE editor: Users Guide, Report 7 (1972), Department of Computer Science, University of Colorado, Boulder, Colorado 80309.

[5] Sale, A. H. J. The Classification of Fortran Statements. Comp. J 14 (1971), 10-12.

Appendix A

```
      SUBROUTINE TYPE(IA1, MM, JTYP)                                    TYP   10
C IDENTIFICATION                                                        TYP   20
C       PROGRAM NAME     TYPE                                           TYP   30
C       SOURCE LANGUAGE ANSI FORTRAN                                    TYP   40
C       DATE             12/01/73                                       TYP   50
C       PROGRAMMER       M. BOLKE                                       TYP   60
C       READER           J.S. DORRENBACHER                              TYP   70
C       UPDATE           1 OF JULY 1974                                 TYP   80
C ABSTRACT                                                              TYP   90
C       THIS PROGRAM CLASSIFIES A CHARACTER OR CHARACTER                TYP  100
C STRING AND ASSIGNS A TYPE CODE TO IT.                                 TYP  110
C METHOD OF USE                                                         TYP  120
C THIS SUBROUTINE IS CALLED BY                                          TYP  130
C       CALL TYPE(IA1, MM, JTYPE)                                       TYP  140
C WHERE                                                                 TYP  150
C IA1 -- VARIABLE CONTAINING THE CHARACTER OR CHARACTERS TO BE          TYP  160
C        CLASSIFIED. (INPUT)                                            TYP  170
C MM  -- FLAG INDICATING HOW IA1 SHOULD BE CLASSIFIED.                  TYP  180
C        MM=0 OR -1. (INPUT)                                            TYP  190
C JTYPE-- INTEGER REPRESENTING THE TYPE CODE. (OUTPUT)                  TYP  200
C        JTYP IS ASSIGNED ACCORDING TO THE FOLLOWING RULE               TYP  210
C  JTYP CHARACTER(S) IN IA1                                             TYP  220
C   1    LETTER                                                         TYP  230
C   2    DIGIT                                                          TYP  240
C   3    END OF STATEMENT MARK (3HEOS)                                  TYP  250
C   4    LEFT PARENTHESIS                                               TYP  260
C   5    RIGHT PARENTHESIS                                              TYP  270
C   6    COMMA                                                          TYP  280
C   7    EQUAL SIGN                                                     TYP  290
C   8    IF MM=0, THEN JTYP=8 IF IA1 IS NOT ONE OF THE ABOVE.           TYP  300
C        IF MM=-1, THEN JTYP=8 IF IA1 IS A PLUS OR MINUS SIGN.          TYP  310
C   9    ASTERISK OR SOLIDUS                                            TYP  320
C  10    PERIOD                                                         TYP  330
C  11    BLANK                                                          TYP  340
C  12    OTHER                                                          TYP  350
C  13    HOLLERITH FLAG (2HH*)                                          TYP  360
C THE CHARACTER TO BE CLASSIFIED IS REPRESENTED IN IA1 IN A1             TYP  370
C FORMAT EXCEPT FOR THE END OF STATEMENT MARK AND THE HOLLERITH         TYP  380
C FLAG.                                                                 TYP  390
C EXAMPLE                                                               TYP  400
C IF IA=1H5 AND MM=-1 THEN CALL TYPE(IA,MM,JTYP) WOULD RETURN           TYP  410
C JTYP=2.                                                               TYP  420
C IF IA=1H$ AND MM=0 THEN JTYP=8 UPON RETURN.                           TYP  430
C PROGRAM CHARACTERISTICS                                               TYP  440
C IA      -- IA1 REPRESENTED AS A LOCAL VARIABLE                        TYP  450
C KLTTR   -- ARRAY CONTAINING ALPHABET IN NORMAL ORDER.  EACH           TYP  460
C            ELEMENT CONTAINS ONE LETTER IN 1H FORMAT                   TYP  470
C KDIG    -- ARRAY CONTAINING DIGITS FROM 0 TO 9. EACH ELEMENT          TYP  480
C            CONTAINS ONE DIGIT IN 1H FORMAT.                           TYP  490
C KSPEC   -- ARRAY CONTAINING SPECIAL CHARACTERS OR SYMBOLS             TYP  500
C            IN H FORMAT.  ORDER OF CHARACTERS IS +, -, *, /, (,        TYP  510
C            ), $, =, BLANK, COMMA, PERIOD, END OF STATEMENT            TYP  520
C            MARK.                                                      TYP  530
C IHOLFG -- HOLLERITH FLAG                                              TYP  540
C THERE ARE NO MACHINE DEPENDENT VARIABLES AND NO MACHINE               TYP  550
```

```
C DEPENDENT CODE IN THIS SUBROUTINE.                                    TYP  560
C MORE EFFICIENT MACHINE DEPENDENT CODE SHOULD BE AVAILABLE AT          TYP  570
C MOST INSTALLATIONS FOR DETERMINING IF A GIVEN CHARACTER IS A          TYP  580
C LETTER OR A NUMBER.                                                   TYP  590
      COMMON /CHRS/ KLTTR(26), KDIG(10), KSPEC(12)                      TYP  600
      DATA IHOLFG /2HH*/                                                TYP  610
      IA = IA1                                                          TYP  620
C CHECK FOR LETTER                                                      TYP  630
      DO 10 I=1,26                                                      TYP  640
        IF (IA.EQ.KLTTR(I)) GO TO 30                                    TYP  650
   10 CONTINUE                                                          TYP  660
C CHECK FOR NUMBER                                                      TYP  670
      DO 20 I=1,10                                                      TYP  680
        IF (IA.EQ.KDIG(I)) GO TO 40                                     TYP  690
   20 CONTINUE                                                          TYP  700
C CHECK FOR END-OF-STATEMENT MARK                                       TYP  710
      IF (IA.NE.KSPEC(12)) GO TO 50                                     TYP  720
      JTYP = 3                                                          TYP  730
      RETURN                                                            TYP  740
C CHARACTER IS A LETTER                                                 TYP  750
   30 JTYP = 1                                                          TYP  760
      RETURN                                                            TYP  770
C CHARACTER IS A NUMBER                                                 TYP  780
   40 JTYP = 2                                                          TYP  790
      RETURN                                                            TYP  800
C CHECK FOR LEFT PARENTHESIS                                            TYP  810
   50 IF (IA.NE.KSPEC(5)) GO TO 60                                      TYP  820
      JTYP = 4                                                          TYP  830
      RETURN                                                            TYP  840
C CHECK FOR RIGHT PARENTHESIS                                           TYP  850
   60 IF (IA.NE.KSPEC(6)) GO TO 70                                      TYP  860
      JTYP = 5                                                          TYP  870
      RETURN                                                            TYP  880
C CHECK FOR COMMA                                                       TYP  890
   70 IF (IA.NE.KSPEC(10)) GO TO 80                                     TYP  900
      JTYP = 6                                                          TYP  910
      RETURN                                                            TYP  920
C CHECK FOR EQUAL SIGN                                                  TYP  930
   80 IF (IA.NE.KSPEC(8)) GO TO 90                                      TYP  940
      JTYP = 7                                                          TYP  950
      RETURN                                                            TYP  960
C CHECK FOR ABRIDGED TABLE                                              TYP  970
   90 IF (MM.NE.0) GO TO 100                                            TYP  980
      JTYP = 8                                                          TYP  990
      RETURN                                                            TYP 1000
C CHECK FOR + OR - SIGN                                                 TYP 1010
  100 IF (IA.NE.KSPEC(1) .AND. IA.NE.KSPEC(2)) GO TO 110                TYP 1020
      JTYP = 8                                                          TYP 1030
      RETURN                                                            TYP 1040
C CHECK FOR * OR / SIGN                                                 TYP 1050
  110 IF (IA.NE.KSPEC(3) .AND. IA.NE.KSPEC(4)) GO TO 120                TYP 1060
      JTYP = 9                                                          TYP 1070
      RETURN                                                            TYP 1080
C CHECK FOR PERIOD                                                      TYP 1090
  120 IF (IA.NE.KSPEC(11)) GO TO 130                                    TYP 1100
```

```
      JTYP = 10                                                        TYP 1110
      RETURN                                                           TYP 1120
C CHECK FOR BLANK                                                      TYP 1130
  130 IF (IA.NE.KSPEC(9)) GO TO 140                                    TYP 1140
      JTYP = 11                                                        TYP 1150
      RETURN                                                           TYP 1160
C CHECK FOR HOLLERITH FLAG.                                            TYP 1170
  140 IF (IA.NE.IHOLFG) GO TO 150                                      TYP 1180
      JTYP = 13                                                        TYP 1190
      RETURN                                                           TYP 1200
C CHARACTER IS OF TYPE OTHER                                           TYP 1210
  150 JTYP = 12                                                        TYP 1220
      RETURN                                                           TYP 1230
      END                                                              TYP 1240
```

Appendix B
BREAK AND APPEND PROCEDURES

When the text of a COMMENT statement extends past KEYS(1) it is broken at the first blank space located at or to the left of KEYS(1) + 1. Decisions concerning the continuation indentation and whether or not to append the next COMMENT line are made by examining as many as necessary, or possible, of the two preceding and two following COMMENT lines. Appending of additional COMMENT lines is terminated when any of the following occur: 1) end of COMMENT block, 2) occurence of blank COMMENT, 3) next COMMENT has greater indentation than current COMMENT, or 4) less than eight characters of the next COMMENT line could be appended to the current COMMENT line. Terminal condition 3) precludes appending the second line of a subparagraph to the first when that form of indentation is used with subparagraphs.

Hints to users:

1. Liberal use of blank COMMENTS is recommended to improve readability and processing of COMMENTS. The blank COMMENTS are removed by the KEYS(8) default value.
2. Generally it is best to separate consecutive subparagraphs with a blank COMMENT. This precludes incorrectly appending subparagraphs together.
3. A single, one line subparagraph should be followed by a blank COMMENT.
4. Use a long text line for COMMENT text, i.e., keep KEYS(2) small.
5. Blank COMMENT statements or an empty COMMENT block delimited by * or + delimiters may be used to separate COMMENT statements. The

former keeps all the COMMENTS in the same block while the later separates the COMMENTS into two blocks.

6. Break and append procedures are not recommended for a COMMENT block that contains an outline format.

The following example contains most of the problem cases and shows the use and misuse of delimiters, blank COMMENTS, and text line length.

Comment block ready to edit.

```
C                    EXAMPLE COMMENT BLOCK
C
C    THE BLANK COMMENT KEEPS A LONG TITLE LINE FROM BEING
C    CONSIDERED AS THE INDENTED FIRST LINE OF THE FOLLOWING
C    PARAGRAPH.  THE BLANK COMMENT FOLLOWING THE SUBPARAGRAPH
C    SERVES A SIMILAR PURPOSE.
C          A. AN ERROR COULD RESULT IF THIS SUBPARAGRAPH BROKEN.
C
C    A DIFFERENT PROBLEM IS ILLUSTRATED BY THESE SUBPARAGRAPHS.
C          A. AN ERROR COULD RESULT IF THIS SUBPARAGRAPH BROKEN.
C
C          B. THE BLANK COMMENT SAVES THE SUBPARAGRAPH FORMAT
C          WHERE THE DELIMITED EMPTY COMMENT BLOCK DOES NOT AS
C          SHOWN HERE.
*
*
C          C. THE SUBPARAGRAPH FORMAT WILL BE LOST.
+
C               HERE THE USE OF A * OR + EMPTY COMMENT BLOCK
C               WOULD HAVE THE SAME RESULT IF KEYS(9).NE.0 BUT
C               WOULD DEPEND ON KEYS(10) IF KEYS(9).EQ.0.
+
$
```

Let KEYS(1)=50, KEYS(2)=10, and KEYS(9)=1. Assume other KEYS are at their default values.

Results of editing are:

```
                EXAMPLE COMMENT BLOCK                                MAN   10
THE BLANK COMMENT KEEPS A LONG TITLE LINE                            MAN   20
FROM BEING CONSIDERED AS THE INDENTED                                MAN   30
FIRST LINE OF THE FOLLOWING PARAGRAPH.                               MAN   40
THE BLANK COMMENT FOLLOWING THE                                      MAN   50
SUBPARAGRAPH SERVES A SIMILAR PURPOSE.                               MAN   60
      A. AN ERROR COULD RESULT IF THIS                               MAN   70
      SUBPARAGRAPH BROKEN.                                           MAN   80
A DIFFERENT PROBLEM IS ILLUSTRATED BY                                MAN   90
THESE SUBPARAGRAPHS.                                                 MAN  100
      A. AN ERROR COULD RESULT IF THIS                               MAN  110
      SUBPARAGRAPH BROKEN.                                           MAN  120
      B. THE BLANK COMMENT SAVES THE                                 MAN  130
      SUBPARAGRAPH FORMAT WHERE THE                                  MAN  140
      DELIMITED EMPTY COMMENT BLOCK DOES                             MAN  150
      NOT AS SHOWN HERE.                                             MAN  160
C. THE SUBPARAGRAPH FORMAT WILL BE LOST.                             MAN  170
HERE THE USE OF A * OR + EMPTY COMMENT                               MAN  180
BLOCK WOULD HAVE THE SAME RESULT IF                                  MAN  190
KEYS(9).NE.0 BUT WOULD DEPEND ON KEYS(10)                            MAN  200
IF KEYS(9).EQ.0.                                                     MAN  210
```

Appendix C
COMMENT TABLE EDITING

COMMENT table editing includes the deletion of indicated card columns, the insertion of a space following an indicated card column, and the breaking and continuation with possible appending of the last table column of the respaced COMMENT table if it exceeds KEYS(1). The broken last column is continued at the indicated card column corrected for revised table spacing. If the following COMMENT table line starts at the indicated continuation indentation it is available to be appended to the broken last column. KEYS(8) is used to edit blank COMMENTS.

COMMENT tables are delimited by the ++ delimiter cards. Immediately following the initial ++ delimiter card must be a data card containing processing information for the COMMENT table. The data card has the following format: columns 1-72 contain +, -, or blank; columns 73-78 are blank; columns 79-80 are blank or contain a right justified card column number corresponding to the card column, uncorrected for added spaces or deleted characters, in which the continuation of the last table column is to be started. When columns 79-80 are blank the last table column is not broken. The last table column is broken at the first blank space located at or to the left of KEYS(1) + 1. The meaning of the characters in columns 1-72 are: -

- delete card column from table,

+ insert space after card column is copied,

blank copy card column as is.

Tables of COMMENTS may contain blocks of code and/or COMMENTS between do not edit delimiters. A non-comment statement not located

between do not edit delimiters will terminate table editing with an error message. If a terminal ++ delimiter card is then encountered, the next card is lost and other errors may be introduced without warning or error messages. The other errors should be restricted to COMMENT statements and COMMENT tables. An error message is provided if a table line exceeds KEYS(1) after editing. KEYS(2) is not used in COMMENT table editing.

Example using table processing to insert and delete blanks.

Input table:

```
++
+++++     +++              -------
C     TIME    SPEED                DISTANCE
C     10.3    54.7                 107.1
C     15.2    52.3                 153.9
++
$
```

Edited table:

```
C           TIME       SPEED      DISTANCE                                  MAN    1(
C           10.3       54.7       107.1                                     MAN    2(
C           15.2       52.3       153.9                                     MAN    3(
```

Example using multiple table editing within one table to delete non-blank character, to align table sections, and to break, continue and append the last column of the table. KEYS(1)=45.

Input table:

```
++
      -
C     IPTR--POINTER TO A-ARRAY.
C     JPTR--POINTER TO B-ARRAY.
++
++
       ++                                                              1
C     A -ARRAY CONTAINING INPUT DATA TO BE PROCESSED.
C        ARRAY IS IN A1 FORMAT.
C     B -ARRAY CONTAINING PROCESSED DATA TO BE OUTPUT.
C        ARRAY IS IN A1 FORMAT.
++
$
```

Edited table:

```
C     IPTR-POINTER TO A-ARRAY.                                  MAN    1
C     JPTR-POINTER TO B-ARRAY.                                  MAN    2
C     A  - ARRAY CONTAINING INPUT DATA TO BE                    MAN    3
C            PROCESSED.                                         MAN    4
C          ARRAY IS IN A1 FORMAT.                               MAN    5
C     B  - ARRAY CONTAINING PROCESSED DATA TO                   MAN    6
C            BE OUTPUT.                                         MAN    7
C          ARRAY IS IN A1 FORMAT.                               MAN    8
```

Appendix D
SECOND LEVEL DEFINITION

The second level definition restrictions on array subscripts of ANSI FORTRAN Standard X3.9-1966, Sections 10.2.8 and B8.1 have not been observed in subprograms COPY, DRICON, DRIVER, LABEL, and LBOCSC. To date, no user has reported any problems in this area. The following changes, line replacements except as noted, will cause second level definition restrictions to be observed:

```
SUBROUTINE COPY
      OUTBUF(I) = A(IP)                                          COP  230

SUBROUTINE DRICON
      IF (KSETB(JP,6).NE.0)    GO TO 330                         DRI 1300
      IF (A(IP).NE.KLTTR(6))   GO TO 190                         DRI 2200
  190 IF (A(IP).NE.KLTTR(18)) GO TO 200                         DRI 2240
  200 IF (A(IP).NE.KLTTR(4))   GO TO 210                        DRI 2280
      B(JP) = A(IP)                                             DRI 2350
      IF (KADVBB(JP).NE.0)     GO TO 330                        DRI 2370
      B(JP) = IBLANK                                            DRI 2400
      IF (KADVBB(JP).NE.0)     GO TO 330                        DRI 2410
      IF (KSETB(JP,5).NE.-1)   GO TO 330                        DRI 2510
      B(JP) = LDSTCK                                            DRI 2520
      IF (K.EQ.10) B(JP) = B(JP) - 1                            DRI 2530

SUBROUTINE DRIVER
      B(J) = A(IP)                                              DRI 1020
      B(JP) = IBLANK                                            DRI 1050
   60 CALL TYPE(A(IP), MM, JTYP)                                DRI 1170
      A(IP) = INST                                              DRI 1300
   90 B(JP) = IBLANK                                            DRI 1420
  100 B(JP) = A(IP)                                             DRI 1450
  120 B(JP) = A(IP)                                             DRI 1600
      B(JP) = IBLANK                                            DRI 1620
```

```
  170 B(JP) = A(IP)                                                  DRI 1850
      IF (KSETB(JP,1).NE.-1) GO TO 470                               DRI 1870
      B(JP) = JJ                                                     DRI 1880
  200 B(JP) = IBLANK                                                 DRI 2060
      B(JP) = A(IP)                                                  DRI 2080
  240 NPOINT = A(IP)                                                 DRI 2260
      CALL TYPE(A(IP), MM, JTYP)                                     DRI 2280
      IF (A(IP).EQ.KLTTR(1) .OR. A(IP).EQ.KLTTR(15))
     * GO TO 260                                                     DRI 2300
  250 B(JP) = NPOINT                                                 DRI 2320
      B(JP) = IBLANK                                                 DRI 2360
      B(JP) = NPOINT                                                 DRI 2380
      B(JP) = A(IP)                                                  DRI 2480
      B(JP) = IBLANK                                                 DRI 2500
  320 B(JP) = IBLANK                                                 DRI 2790
      B(JP) = A(IP)                                                  DRI 2910
      NAMES(LL) = A(IP)                                              DRI 2920
      IF (A(IP).EQ.KSPEC(5)) GO TO 60                                DRI 2950
  380 IF (A(IP).NE.KSPEC(4)) GO TO 100                               DRI 3120
      NCHAR = A(IP)                                                  DRI 3200
  410 B(JP) = A(IP)                                                  DRI 3300
      B(JP) = A(IP)                                                  DRI 3330
      B(JP) = KSPEC(9)                                               DRI 3370
      B(JP) = A(IP)                                                  DRI 3430
      IF (A(IP).NE.IBLANK) GO TO 60                                  DRI 3470
```

Replace DRI 1030, 1430, 1460, 1610, 1630, 2070, 2090, 2330, 2370, 2390, 2490, 2510, 2800, 2930, 3320, 3350, 3380, and 3440 by

```
      IF (KADVBB(JP).NE.0) GO TO 470
```

```
SUBROUTINE LABEL

    * IP, JP                                              LAB  890
 20   IF (KADVBA(IPNTRA).GT.0) GO TO 210                  LAB 1210
      IF (KADVBB(IPNTRB).GT.0) GO TO 210                  LAB 1250
      IF (KADVNA(IPNTRA).LT.0) GO TO 210                  LAB 1310
 60   IF (KDECBA(IPNTRA).NE.0) GO TO 210                  LAB 1360
      IF (KSETB(IPNTRB,3).GT.0) GO TO 210                 LAB 1410

Insert after LAB 1060
      IPNTRA = IP
      IPNTRB = JP

FUNCTION LBOCSC
     * IP, JP                                             LBO  130
      IF (KSETA(IPNTRA,ISTART).NE.0) GO TO 50             LBO  190
      IF (KDECBA(IPNTRA).NE.0) GO TO 50                   LBO  330
```