STYLE Editor: User's Guide

by

Dorothy E. Lang*
Department of Computer Science
University of Colorado
Boulder, CO 80302

Report #CU-CS-007-72 Nov. 1972

* This work supported in part by NSF Grant GJ 660

STYLE Editor: User's Guide

by

Dorothy E. Lang

ABSTRACT

STYLE is a two-pass special purpose text editor, specifically designed to reformat FORTRAN source code. STYLE accepts FORTRAN programs set up in logical segments as its data input, and produces as its output the equivalent program edited into a specific stylized format according to certain parameters. STYLE itself is a FORTRAN subprogram system following all ANSI standards and conventions. Some provisions of STYLE are sequencing, editing of statement labels, indentation of DO-loops, spacing of text, restriction of text to column boundary, and editing of comment statements. Usage of the STYLE system is discussed in this paper.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Today with the large number of algorithms being published in journals and with recent emphasis on centralized mathematical subroutine libraries and transportability, it becomes increasingly important to understand how a program functions within a reasonably short period of time. Often, however, readability of programs decreases as development increases - particularly with large systems of programs involving many programmers each having a chance to supply some of their own logic and programming tricks. How then, does one reestablish "readability" so that the code is legible to a potential user? It appears that standardization not only of syntax, but also of style is a well warranted consideration in the computing field.

STYLE is an editing system which attempts to construct "readable" programs for the FORTRAN language. Its purpose is to transform a given FORTRAN source program into a functionally equivalent FORTRAN program which has been edited into a specific stylized format. The STYLE system fulfills the minimum standards set forth for FORTRAN in the Communications of the ACM (Association for Computing Machinery), but at the same time gives the user considerably more freedom than implied by the CACM policy (see [2]). STYLE itself is written in ANSI FORTRAN and accepts a fixed set of parameters (editing features) which have a wide range of values. The default value of each parameter coincides with CACM preference. When it was thought to complicate either the logic or the simplicity in using the system, some potential parameters became fixed editing features rather than user specified. Thus some editing features of STYLE, particularly spacing, are not under user control.

Also, in recognition of the many dialects of FORTRAN, STYLE handles some non-ANSI FORTRAN statements.

The general structure of STYLE allows editing of statement labels, indentation of statements within the range of a DO statement, editing of comment statements, sequencing, restriction of right-hand column boundary, spacing of text, etc. A brief description of some of the general system features is given below. For a complete description refer to the instant reference manual (Appendix C).

1. user specified output format of program providing a flexible system (see Appendix C, Tables 1, 2)
2. recognition of some non-ANSI FORTRAN (See Appendix C, Table 3, STYLE does not edit these statements)
3. extensive error diagnostics (see Appendix C, Table 4) including informative diagnostics when system error conditions are detected (see Appendix C, Table 5)
4. standard set of editing features which can easily be extended to meet individual requirements
5. repositioning of FORMAT statements to beginning of subprogram unit if desired
6. production of new edited source, if desired
7. deletion of unreferenced statement labels
8. recognition of assembly language subroutine (no processing)

A complete example of the capabilities of STYLE appears in Appendix A. Some short examples appear below:

<u>example 1 - spacing</u>

before:

```
      DIMENSIONA(80),B(4,4),C(20)
```

after:

```
      DIMENSION A(80), B(4,4), C(20)
```

example 2 - indentation of blocks, spacing

before:

```
      DO17I
17    K=999-MOD(I,10)/10+(6-K3)
```

after:

```
      DO 17 I=1,N
        K = 999 - MOD(I,10)/10 + (6-K3)
   17 CONTINUE
```

example 3 - meet CACM standards

before:

```
C*****************************************************************
C
C     PRINT MESSAGE
C
C*****************************************************************
      WRITE(7,1)I,(LINE(J),J=1,N)
1     FORMAT(* ERROR*I5* OCCURRED ON THE FOLLOWING LINES-*/25I5)
```

after:

```
C PRINT MESSAGE
       WRITE(7,10) I, (LINE(J),J=1,N)
     10 FORMAT(6H ERROR, I5, 23H OCCURRED ON THE FOLLOW,
       * 10HING LINES-/25I5)
```

In order for STYLE to perform its task, the user must define to the system his resultant output format. This is done via a set of parameters referred to as keys. These keys may be thought of as commands to STYLE indicating how one would want his program to look. The keys provide specifications on handling such details as comment statements, DO-loops, statement labels, and margins. For example, if a user wishes statement labels to monotonically increase in steps of 10, he would set KEYS(4) to 10.

Along with specifying the keys to the system, the user must also provide location information about his input program and desired output program. This is done by assigning device unit numbers and associating them with the appropriate files. For example, if the input program is on punched cards, and the installation standard device unit for input cards is 5, the user would specify the integer number 5 to STYLE as the

input device. Similarly, if a new punched deck were desired and the standard device unit for punched output is 7, the user would specify the integer number 7 as the punch file.

STYLE is a self-contained subprogram unit and runs in a batch environment. A single call to STYLE must provide all the format specifications and files desired. Generally, the user will write a FORTRAN driver (main program) calling STYLE once for every program to be reformatted. The user need not supply the format specifications (defaults will be used), but must provide the file information and some buffer size information. In addition, a -999 statement as the last data card is used to indicate end of file.

It is expected that the average user of STYLE will find all the facilities that he needs included in the STYLE system. However, the user with source statements unknown to the STYLE system (FORTRAN dialects) or with a specialized formatting structure not covered in STYLE, may find the extensibility of the STYLE system a handy and powerful tool. On the other hand, the user with close to ANSI FORTRAN programs may wish to reduce the standard feature set, and thereby reduce core and time requirements.

CHAPTER 2

CALLING SEQUENCE

STYLE is FORTRAN callable as follows:

```
CALL STYLE(KEYS, IDENT, N, IDEV, IERROR)
```

where:

KEYS is an 11 word integer array defining the editing options.

IDENT is a 3 word integer array defining the identifier to associate with sequencing of program.

N is the maximum number of continuation cards the input program contains.

IDEV is a 5 word array defining the device unit numbers for all files.

IERROR is an error return flag, returning nonzero value if an error was detected in the run, otherwise returns zero.

(all parameters, including alphabetic data, are treated as integer values)

The eleven keys control the editing and thus control the output format of the program. A zero value for any key will result in its corresponding default value being used. The default values correspond to the CACM specifications for FORTRAN algorithms [2,676]. The keys have the following meaning and implications:

KEYS(1) - rightmost position (column) that the edited text may appear.
minimum - 7
maximum - 72
default - 64

KEYS(2) - left margin for text in a comment statement. If the text surpasses KEYS(1), it is split into as many comment statements as needed upon output.
minimum - 2
maximum - KEYS(1)
default - 3
no edit - any negative value

KEYS(3) - position (column) to right justify edited label.
minimum - 1
maximum - 5
default - 5

KEYS(4) - increment value for edited labels - also initial value can be any positive integer $\leq$ 99999.
minimum - 1
maximum - 99999
default - 10

KEYS(5) - indentation for text on a continuation.
minimum - 1
maximum - KEYS(1) - 6
default - 1

KEYS(6) - increment value for sequencing in positions 77-80 (4 positions) - also initial value.
minimum - 1
maximum - 9999
default - 10

KEYS(7) - indentation for text within DO-loops.
minimum - 1
maximum - KEYS(1) - 6
default - 2

KEYS(8) - comment statements that are blank.
delete - 0
leave - 1
default - 0

KEYS(9) - comment statements containing all asterisks.
delete - 0
leave - 1
default - 0

KEYS(10) - table of old vs. new statement labels.
do not print - 0
print - 1
default - 0
if specified, a table will be printed after each subprogram unit of the input program.

KEYS(11) - break character used to delimit Hollerith constants (examples - ' or *), left justified in the word.
default - 0 no character
The Hollerith constant delimited by said character will be changed to the standard H specification.

In specifying the keys, it is important not to have them overlap in such a way as to contradict each other. Few checks are made to detect

incompatible key specifications. STYLE does check the two keys concerned with statement labels - KEYS(3) and KEYS(4). Labels can only be five digits. If the increment value times the number of labels is greater than 99999, the increment value is decremented by one until all labels will fit. The margin specification, KEYS(3), is checked to see if the maximum label will fit within that space. If not, the default is set to 5. If a change is made, that fact appears as a message on the output listing. No other checks are made.

The array, IDENT, is associated with sequencing. The user specifies a 3-character identifier. This identifier appears in positions 73-75 of every subprogram unit. Note that KEYS(6) specifies sequencing in positions 77-80. Character position 76 is reserved for a unique alphanumeric character for each subprogram unit beginning with A, ending with 9. The characters will repeat beginning with the 37th subprogram unit. The provision allows for sorting on positions 76-80.

The array IDEV allows the user to define what files he will be using and their associated device units. Input, output, and one intermediate file must be defined. The punch and second scratch file are optional. The array values are positive integer numbers and have the following meanings:

```
IDEV(1) - input, FORTRAN source program
IDEV(2) - output, listing
IDEV(3) - punch (optional), edited FORTRAN program
IDEV(4) - scratch file 1, intermediate text
IDEV(5) - scratch file 2 (optional), FORMAT statements
```

The two optional files are set to zero if they are not desired. Setting IDEV(5) = 0, results in no repositioning of FORMAT statements.

The parameter N is a non-negative integer less than 19. It specifies the maximum number of continuation cards that the input program contains. This value is used to calculate the maximum buffer length needed for a single run of STYLE. An incorrect value causes a system error (see chapter 4). Since the buffers can be dynamic, this value increases or decreases the machine space requirements necessary to execute STYLE (see chapter 5).

IERROR is an error return flag which may or may not be used by the user upon return from the STYLE system. It could be used to release the punch output file if in error, for example. The values of IERROR are:

IERROR = 0, no errors detected by system

IERROR $\neq$ 0, error detected by system

CHAPTER 3

RESTRICTIONS

Several restrictions are made in STYLE to simplify the use of the system. They can be classified into four areas:

1) non-ANSI FORTRAN
2) semantics
3) spacing
4) end-of-file (EOF) test

Each will be discussed briefly.

3.1. Syntax and non-ANSI FORTRAN

STYLE performs no syntax check in two commonly misused areas:

1) subscript expressions
2) Hollerith constant in function definitions

The effect for subscripts is that any subscript expression is acceptable, including subscripted subscripts.

STYLE will recognize and process some common ANSI violations. These are:

1) special character delimiter for string data rather than H specification (this is not automatic; it must be specified via a key).
2) character data specifications such as R - right justified, zero fill.

Lastly, STYLE will recognize but not process non-standard statements such as the OVERLAY statement. A complete list appears in the

instant reference manual, Appendix C. This set can be easily tailored by the user to meet his individual requirements.

The reasons for adopting such restrictions are associated with the many problems in performing a valid syntax check on a string of data. Aside from the fact that the FORTRAN grammar is not easy to analyze, the problem is compounded by the many existing dialects of FORTRAN and by the lack of any standard of the ANSI X3J3 (FORTRAN) committee in certain areas. How then, should STYLE compensate for the existence of dialect FORTRAN and still be a viable system for all users? To take all such considerations into the design would involve an almost super-human effort. Allowing that some non-ANSI FORTRAN should be recognized, it was, however, considered infeasible from a practical design standpoint to let the user specify his non-standard FORTRAN. Thus, STYLE is selective in its processing of non-ANSI FORTRAN.

3.2. Syntax vs. Semantics

Due to the complexities and syntax of FORTRAN itself, whenever ambiguity arises due to a semantic issue, the ambiguity is allowed to remain unresolved and is assumed correct. The additional time and space requirements necessary to build tables and perform table look-up to handle the semantics was considered unwarranted in view of the fact that the input program to STYLE should be a running program that is already syntactically correct. It is therefore not the purpose of STYLE to perform a full syntactical analysis on the input. It is only necessary to analyze the string in enough detail to accomplish the editing specified by the user. It is for this reason

that such ambiguities (mixed-mode arithmetic is one), if they exist, are tolerated.

3.3. Spacing

Spacing around operators and delimiters is restricted to a schema used in STYLE and is not user controllable. The feeling here is that normal English spacing conventions should be sufficient, a single space after a comma marking off a group, for example. Spacing of operators relies on basically the same grouping principle - the operations performed first are grouped closest together. Most mathematical texts print expressions in this manner. (In fact, most mathematical texts omit the multiplication symbol altogether.) For example, expressions such as max $Z = x_1 + 3x_2$ used in maximization of functions or $Ax = b$ in matrix equations, are commonly seen. A simple hierarchy of operations scheme was adopted in STYLE based upon this idea. It uses the principle that increasing hierarchy implies decreasing spacing. The scheme developed follows:

operator(↑precedence)	spacing
+, -, =, .AND., .OR.	single space before and after
*, **, /, relationals[1]	no space before or after
parentheticals	no space except before and after logical operators[2]

[1] relationals - .LT., .LE., .EQ., .NE., .GE., .GT.

[2] logical operators - .AND., .OR.

3.4. End of File (EOF)

STYLE uses a flag system to signal end of data. It is in the form of a -999 in columns 1-4 on the last data card (behind all subprograms of the source deck). This allows testing of the input device for end of file or end of information in a machine independent manner.

This method presents no problems as long as the input source program is on cards. However, insertion of the -999 card can be annoying especially if the program is stored on some other device, such as magnetic tape or disk. Those users having a specialized end of file test available to them may wish to incorporate such a test, avoiding the -999 card altogether. This can be done most simply by writing an integer function EOFT which returns 1 if end of file is detected, 0 otherwise and replacing this function with the existing one. (This will not work for an END= clause on the READ statement, only for EOF subroutines or functions.)

CHAPTER 4

DIAGNOSTICS AND ERROR PROCESSING

The STYLE editor produces two types of error diagnostics. System errors refer to conditions causing the STYLE system to abort, such as an incorrect specification of a key. There is a corrective procedure for each of these errors.

The second type of error diagnostic concerns itself with the condition of the source statements. These are generally syntax errors.

4.1. System Errors

STYLE keeps careful check on possible table or buffer overflow, and the boundary restrictions set up by the user. If an overflow or boundary violation is detected, STYLE prints out an informative message of the condition and suggests possible corrective procedure. There are six such system error conditions. All six cause immediate termination of the run.

1. BUFFER OVERFLOW, CHECK MAX. NO. OF CONTINUATIONS.

   This message means the input routine has detected more continuation statements than the input parameter N specified, and cannot add any more continuations to the input buffer. This condition stops processing with a STOP 1. The user must rerun the job (and respecify N).

2. OVERFLOW OF LABELS, INCREASE DIMENSION OF LDEF AND LREF.

   This message implies that there are more statement labels in a single subprogram than the maximum allowed for in the tables. The condition stops processing with a STOP 2. Correction requires increasing the dimension of arrays LDEF and LREF (which are the tables of label definitions and label references respectively),

changing the variable MAX in SUBROUTINE INIT to this new value, and recompiling the system.

3. DO STATEMENTS NESTED TOO DEEPLY, INCREASE FIRST DIMENSION OF LAB.

   This message implies that the DO blocks in this subprogram unit are nested too deeply. The table of DO-terminator labels has overflowed. The condition stops processing with a STOP 3. The array LAB is used for the table and is 2-dimensional (two fields per entry). It is only necessary then to increase the first dimension to increase table capacity. The variable MAXI in SUBROUTINE INIT must be set to this new value. The system must be recompiled.

4. ARRAY OVERFLOW, INCREASE DIMENSION OF LENGTH.

   Array LENGTH is the working buffer. Its size depends upon the maximum length of a single token. It is presently set at 30, and in normal circumstances it should not overflow. However, in non-ANSI environments it is conceivable to have larger tokens. Some possible causes of an overflow are:

   1. subscripted subscripts.
      example: (NAME(I,J),NAME(I+1,J),NAME(I+2,J))

   2. more than three dimensions
      example: 5 dimensions with each subscript a 6 character variable yields 36 characters

   3. variable names of 6 characters in combination with either of the above.

   The condition halts processing on a STOP 4. Correction requires increasing the dimension of LENGTH, changing the variable LEX in SUBROUTINE INIT to this value, and recompiling the system.

5. N (NUMBER OF CONTINUATION CARDS) INCORRECTLY SPECIFIED. MUST BE NON-NEGATIVE INTEGER LESS THAN 19.

   The parameter N has not been passed correctly to STYLE. The condition halts processing on a STOP 5. Correction requires checking of data cards and resubmitting the job.

6. HOLLERITH CONSTANT IN DATA OR CALL TOO LONG FOR SPECIFIED COLUMN BOUNDARY i
   where i is the boundary specified by KEYS(1)

   This message implies that a Hollerith argument is too long for the number of characters that can fit on a line. For example, if a user specifies the rightmost margin for text as column 50 (KEYS(1) = 50), then no

Hollerith argument in a subroutine call or data statement can have more than 41 (50-6=44, minus 3 more for the count and the H) characters. The constant cannot be split across to the next continuation card because blanks are meaningful. The constant cannot be split into more than one (as is done with FORMAT statements) because the argument count would no longer be correct. The result is a halt in processing on a STOP 6 condition. The user must rerun the job (and respecify KEYS(1) to accommodate the mishap).

4.2. Source Statement Errors

During the processing of the input program there is always the possibility of error in a statement. There are four classes of errors that STYLE will recognize - two considered fatal, two non-fatal. Fatal errors are flagged on the output listing directly under the statement containing the error. For example:

```
   10 IF(IABS(I) 20, 30, 20
***** ERROR CODE 28 *****
```

Error code 28 is a syntax error of punctuation, probably a missing right parenthesis,), or an extra left parenthesis, (. Fatal errors also suppress any further output of the new text (normally a new punched card deck).

The example above is a syntax error. Another type of fatal error is order of the FORTRAN statement types themselves. An arithmetic IF statement not followed by a labelled statement is an example of this type of error.

STYLE has a list of fifty fatal error codes. This list appears in the instant reference manual, Appendix C, Table 4. The first eight are sequence errors and the remaining (except #50) are syntax errors.

All recognized non-ANSI statements and violations are considered non-fatal errors. These statements are not flagged on the listing. Rather, a list of the statement numbers where the errors occur is printed at the end of the subprogram unit. The second type of non-fatal error can occur if a statement, after being edited, will not fit in the output buffer. These statements appear on the listing as they appeared on input, and again at the end of the subprogram unit, a list containing their statement numbers will be printed.

CHAPTER 5

SYSTEM REQUIREMENTS

5.1. Storage

The STYLE system performs psuedo dynamic storage allocation on its input and output buffers at execution time. A position in the buffer is managed dynamically; however, the buffer resides as the last item in blank common. This allows the buffer size to grow without redimensioning on some systems. (Burroughs being the notable exception). However, it must be kept in mind that generally requests for memory are necessary and the user must be sure that he has allotted himself enough. The advantage: the amount of memory needed is dependent on the job to be done, not on the STYLE system itself. In environments where job priorities depend on memory requests, this may make the difference between quick or slow turn around.

From a practical standpoint, however, always requesting the amount of memory needed to run a job is not desirable. Consequently, it is recommended that the STYLE system be initially set up to handle a normal program process (whatever that may be). For example, at the University of Colorado, the STYLE system has been set up to handle a maximum of two continuation cards, 150 statement labels per subprogram unit, punched output, and FORMAT statements pressed to the beginning of a subprogram unit (requires an additional file) as a standard. This system requires 23400_8 words to run on the CDC - 6400 with all file buffers (in this case 6) set at 100_8 words. For systems having I/Ø buffers as part of the operating system rather than part of the running program, the STYLE system

requires 600_8 words less, plus whatever conversion factor applies. The user with a more specialized program, requiring more continuation cards than two for example, must provide enough storage to run - an additional 132 (66*2) words per additional continuation. The user requiring more than 150 statement labels will have to go back to the source and make changes (as indicated in chapter 4) to the system, increasing the storage requirement by the appropriate amount.

The following set of memory requirements applies to the CDC 6400 system at the University of Colorado with the standard setup described above.

University of Washington RUN compiler:

compilation - 36400_8

load - 32100_8 (relocatable binary)

run - 22100_8 (not including driver)

CDC FTN 3.0 compiler:

compilation - 43600_8

5.2. Time

The time required to execute a STYLE run is dependent on the complexity and length of the input source program, and the speed of the machine in use, and the accessibility of scratch files. STYLE is I/Ø bound, and the access time of peripheral storage devices (disks, tapes, or drums) is an important factor in the real-time requirements.

The object deck of STYLE has been obtained from the FORTRAN compiler RUN (University of Washington version) which does not generate particularly efficient object code. However, the following statistics from the University of Colorado's CDC 6400 should help give an

estimate of the time required to run STYLE.

Execution time of example in seconds (Appendix A, 49 lines)	3.46

CHAPTER 6

HELPFUL HINTS

The STYLE system is limited in the sense that it expects a working program as its input. This assumption can cause unexpected results on occasion if the input program is not in proper order. As long as the input source program is ANSI FORTRAN there should be no problems. Use of non-standard statements results either in error or in recognition but no editing. One must keep in mind that STYLE is not a compiler, merely a specialized editor that has been designed to be compatible with ANSI FORTRAN.

The purpose of this section is to forewarn users of the possible consequences of non-ANSI FORTRAN, and to inform users in general of possible problems. It is recommended that in all cases, the user consult his general listing to assure that everything has been processed correctly. The reader should keep in mind that in the following discussion, the phrase 'fatal error' implies suppression of the new deck output (usually a punched deck), but not halting of the run.

1. The STYLE system depends on recognizing an END card to initiate its second pass. Missing END statements cause subsequent subprogram units to list out along with the unit missing the statement. In the case of the last subprogram unit missing an END statement, the result is no listing of that unit at all!

2. Non-ANSI statements are not processed by STYLE. Since many of these non-standard statements are some form of I/Ø statement, the

corresponding FORMAT statement becomes unreferenced. The general policy of STYLE is to delete unreferenced statement labels. Since FORMAT statements always require a label, STYLE supplies a new label and flags the statement in possible error. This avoids duplication of labels and allows legitimately unreferenced FORMATs to either remain or be pulled depending on the circumstances. In any case, the user must realize that this is not a fatal error, only a warning. Therefore, the new deck output is not suppressed. To assure a valid punched deck, the user had best consult his listing.

3. FORMAT statements using a special character delimiter rather than the H specification for Hollerith constants are flagged non-ANSI, but appear corrected (to ANSI specification) on both the listing and new deck output. This is also true of missing field separators (slash or comma) after Hollerith constants.

4. It is possible for a statement to overflow the output buffer after it has been edited, but because of its compact form not to have overflowed the input buffer. STYLE considers this a non-fatal error. Consequently, the new deck output is not suppressed. The statement will appear exactly as it did upon input both on the listing and the new deck output. A list of line numbers will be printed at the end of the subprogram unit where this phenomenon has occurred. Again, it is recommended that the user consult his listing for any such occurrences. Unless the buffer size is already at maximum (19 continuations) the buffers can be lengthened and the job rerun.

5. It is possible for a statement to fulfill the spacing specifications, still fit in the output buffer, but upon editing the statement for output

have it surpass the 19 continuation card limit of FORTRAN. This is considered a fatal error. The user has no choice but to split the statement and resubmit the job, or to respecify the right margin boundary KEYS(1).

6. All information appearing in columns 73-80 is lost upon input. This applies to comment statements.

7. Comment statements that violate the right most margin for text parameter (KEYS(1)) are split into two or more comment statements. A possible consequence of this is:

```
C SUBROUTINE LONGST IDENTIFIES ST. TYPES THAT ARE MORE THAN 6 CHAR.
C IN LENGTH.  IF IT FINDS A MATCH, IT SETS THE PROPER ITYPE FOR SE-
C QUENCE CHECKING AND KSTATE FOR SYNTAX ANALYSIS.
```

getting converted to the following when KEYS(1) = 60:

```
C SUBROUTINE LONGST IDENTIFIES ST. TYPES THAT ARE MORE THAN
C 6 CHAR.
C IN LENGTH.  IF IT FINDS A MATCH, IT SETS THE PROPER ITYPE
C FOR SE-
C QUENCE CHECKING AND KSTATE FOR SYNTAX ANALYSIS.
```

Care should be exercised in setting this parameter.

8. Immediately upon detection of the first error, the new deck output file is suppressed. However, the file is written until the error is detected. Therefore a partial deck will be produced. The listing should be consulted to determine if this is a good deck. (Those users having operating systems that allow rewinding of the system punch file may wish to rewind and endfile the file by testing IERROR upon exit from STYLE.

9. It is a good idea to get a table of old vs new statement labels with the listing - especially for non-standard FORTRAN programs. In the old label table FORMATs will appear as negative numbers, undefined labels as 100000 +the undefined label (at the end of the list). Unreferenced labels will appear as 100000 in the new label table.

10. Specification of KEYS(11) - break character for Hollerith constants - can never hurt. In fact, a user unaware of non-standard Hollerith constants in his program will have all such statements flagged as syntactically incorrect unless the break character is known to the system via KEYS(11). The non-standard Hollerith is changed to the standard H specification format.

11. Incorrect specification of the parameter N (upper bound on the number of continuation cards) can cause several different errors:

real number	- STYLE will not pass the parameter passing stage, N must be integer between 0 and 19.
negative integer	- causes incorrect calculation of buffer, results in system error (see Chapter 5).
underestimate	- eventually causes overflow of buffer, results in system error (see Chapter 5).
overestimate	- no problem unless over 19. Number over 19 causes incorrect calculation of buffer, results in system error (see Chapter 5).

12. ANSI standards provide for a 6 character identifier (variable name). Variable names with greater than 6 characters will be flagged in error.

REFERENCES

(1) USA Standard FORTRAN, United States of America Standards Institute (now American National Standards Institute) New York (1966) 36p

(2) CACM, Vol. 14, No. 10 (1971) p 676, "Algorithms Policy"

(3) Lang, D. E.; STYLE: An Exercise in Transforming FORTRAN Programs Into a Stylized Text; M. S. Thesis, University of Colorado (1972).

APPENDIX A

EXAMPLE OF A STYLE RUN

A listing of the original subroutine used for this example is shown in Figure 1. It is hard to read in several respects:

1) text of comment begins in column 7 as DO statements;
2) existence of nested but not indented DO-loops;
3) statement labels appearing in different columns.

Also, the FORMAT statement does not conform to ANSI standard.

The desired structure for this routine follows:

1) FORMAT statements repositioned.
2) no text past column 50.
3) right margin for labels column 3.
4) increment labels by 5.
5) sequence by 10.
6) indent DO-loops 3 spaces.
7) indent continuation cards 2 spaces.
8) text of comments in column 3.
9) delete blank comments.
10) identifier - TST.
11) punch a new deck.

The necessary KEYS to specify are numbers 1, 3, 4, 5, and 7. KEYS(s), for comment text, has a default of 3, KEYS(6), sequence increment, has a default of 10 as specified. KEYS(8) and KEYS(9) delete all blank and all asterisk comment statements. Following the suggestions of chapter 6, KEYS(10) and KEYS(11) should be specified. KEYS(10) is set to print the table of old vs new labels; KEYS(11) is the Hollerith constant delimiter. The KEYS, then, would be:

```
KEYS(1)  =  50
KEYS(3)  =   3
KEYS(4)  =   5
KEYS(5)  =   2
KEYS(7)  =   3
KEYS(10) =   1
KEYS(11) =   *
```

The KEYS not set will be set to their corresponding defaults within STYLE.

```
      SUBROUTINE IMPRVZ(N,NX,A,LU,B,X,DIGITS,IPS,R,DX)
C
C          A SUBROUTINE WHICH IMPROVES THE SOLUTION FROM SOLVE
C
C     N  THE ORDER OF A
C     A  THE MULTIPLYING MATRIX
C     LU SAVED MULTIPLYERS
C     B  THE CONSTANT VECTOR
C     X  THE SOLUTION
C     DIGITS  AN APPROXIMATION TO THE NUMBER OF CORRECT DIGITS IN X
C
      DOUBLE TA,TX,TB
      DOUBLE SUM
      DIMENSION A(NX,1),LU(NX,1),B(1),X(1),R(1),DX(1),IPS(1)
      REAL LU
      EPS=1630 7777 7777 7777 7777B
      ITMAX=50
C     EPS  AND  ITMAX  ARE MACHINE DEPENDENT
      XNORM=0.
      DO 1 I=1,N
    1 XNORM=AMAX1(XNORM,ABS(X(I)))
      IF(XNORM.GT.0.) GO TO  3
      DIGITS=-ALOG10(EPS)
    2 RETURN
    3 DO 9 ITER=1,ITMAX
C
      DO 5 I=1,N
      SUM=0.
      DO 4 J=1,N
      TA=A(I,J)
      TX=X(J)
    4 SUM=SUM+TA*TX
      TB=B(I)
      SUM=TB-SUM
    5 R(I)=SUM
      CALL SOLVEZ(N,NX,LU,R,DX,IPS)
      DXNORM=0.
      DO 6 I=1,N
      T=X(I)
      X(I)=X(I)+DX(I)
    6 DXNORM=AMAX1(DXNORM,ABS(X(I)-T))
      IF(ITER.GT.1) GO TO 8
      DIGITS=-ALOG10(AMAX1(DXNORM/XNORM,EPS))
    8 IF(DXNORM.LE.EPS*XNORM) RETURN
    9 CONTINUE
C     ITERATION DID NOT CONVERGE
      WRITE(7,99)
   99 FORMAT(' ITERATION DID NOT CONVERGE')
      RETURN
      END
```

FIGURE 1

The 3 character identifier, TST, is set as follows:

```
IDENT(1)  =  T
IDENT(2)  =  S
IDENT(3)  =  T
```

The N parameter is set to zero as there are no statements in this routine which require continuations (refer to Figure 1).

Lastly, the device units for the files used in the run must be specified.

STYLE requires one scratch file for intermediate text. Repositioning of FORMATs requires an additional scratch file. These will be assigned to units 1 and 2, respectively. For purposes here input will be device unit 5, output device unit 6, and punch device unit 7. (It is understood the user is responsible for requesting these files as necessary for his operating system.) In this example the device units are set as follows:

```
IDEV(1)  =  5
IDEV(2)  =  6
IDEV(3)  =  7
IDEV(4)  =  1
IDEV(5)  =  2
```

The parameter IERROR need not be set upon calling STYLE.

The output appears as Figure 2.

```
S T Y L E   VERSION 1.0 AUG 72

   1                      SUBROUTINE IMPROVE(N, NX, A, LU, B, X,
                         .  DIGITS, IPS, R, DX)                          TSTA  10
   2                C                                                    TSTA  20
   3                C A SUBROUTINE WHICH IMPROVES THE SOLUTION FROM      TSTA  30
   4                C   SOLVE                                            TSTA  40
   5                C                                                    TSTA  50
   6                C N  THE ORDER OF A                                  TSTA  60
   7                C A  THE MULTIPLYING MATRIX                          TSTA  70
   8                C LU SAVED MULTIPLYERS                               TSTA  80
   9                C B  THE CONSTANT VECTOR                             TSTA  90
  10                C X  THE SOLUTION                                    TSTA 100
  11                C DIGITS  AN APPROXIMATION TO THE NUMBER OF          TSTA 110
  12                C   CORRECT DIGITS IN X                              TSTA 120
  13                C                                                    TSTA 130
  14                      DOUBLE TA,TX,TB                                TSTA 140
***** ERROR CODE 15 *****                                               TSTA 150
  15                      DOUBLE SUM
***** ERROR CODE 15 *****                                               TSTA 160
  16                      DIMENSION A(NX,1), LU(NX,1), B(1), X(1),      TSTA 170
                         .  R(1), DX(1), IPS(1)                         TSTA 180
  17                      REAL LU                                       TSTA 190
  18                999   FORMAT(27H ITERATION DID NOT CONVERGE)        TSTA 200
  19                      EPS=1640 7777 7777 7777 7777B                 TSTA 210
***** ERROR CODE 30 *****
  20                      ITMAX = 30
  21                C EPS  AND  ITMAX  ARE MACHINE DEPENDENT            TSTA 220
  22                      XNORM = 0.                                    TSTA 230
  23                      DO 5 I=1,N                                    TSTA 240
  24                         XNORM = AMAX1(XNORM,ABS(X(I)))             TSTA 250
  25                 5    CONTINUE                                      TSTA 260
  26                      IF (XNORM.GT.0.) GO TO 10                     TSTA 270
  27                      DIGITS = -ALOG10(EPS)                         TSTA 280
  28                      RETURN                                        TSTA 290
  29                 10   DO 35 ITER=1,ITMAX                            TSTA 300
  30                C                                                   TSTA 310
  31                         DO 20 I=1,N                              TSTA 320
  32                            SUM = 0.                              TSTA 330
  33                            DO 15 J=1,N                           TSTA 340
  34                               TA = A(I,J)                        TSTA 350
  35                               TX = X(J)                          TSTA 360
  36                            SUM = SUM + TA*TX                     TSTA 370
  37                 15         CONTINUE                              TSTA 380
  38                            TB = B(I)                             TSTA 390
  39                            SUM = TB - SUM                        TSTA 400
  40                            R(I) = SUM                            TSTA 410
  41                 20      CONTINUE                                 TSTA 420
  42                         CALL SOLVEZ(N, NX, LU, R, DX, IPS)       TSTA 430
  43                         DXNORM = 0.                              TSTA 440
  44                         DO 25 I=1,N                              TSTA 450
  45                            T = X(I)                              TSTA 460
  46                            X(I) = X(I) + DX(I)                   TSTA 470
  47                            DXNORM = AMAX1(DXNORM,ABS(X(I)-T))    TSTA 480
  48                 25      CONTINUE                                 TSTA 490
  49                         IF (ITER.GT.1) GO TO 30                  TSTA 500
  50                         DIGITS = -ALOG10(AMAX1(DXNORM/XNORM,EPS)) TSTA 510
  51                 30      IF (DXNORM.LE.EPS*XNORM) RETURN          TSTA 520
  52                 35   CONTINUE                                    TSTA 530
  53                C ITERATION DID NOT CONVERGE                      TSTA 540
  54                      WRITE (7,999)                               TSTA 550
  55                      RETURN                                      TSTA 560
  56                      END                                         TSTA 570
                                                                      TSTA 580
```

FIGURE 2

```
S T Y L E   VERSION 1.0 AUG 72


LABELS-      OLD       NEW
                1    100000
          -100000         5
                2    100010
                3        10
                4    100015
          -100000        15
                5    100020
          -100000        20
                6    100025
          -100000        25
                8        30
                9        35
              -99       999

TOTAL NUMBER OF ERRORS (EXCLUDING ANSI VIOLATIONS) IS   3

TOTAL NUMBER OF ANSI VIOLATIONS IS   1
(STRING DATA IN FORMATS HAVE BEEN CHANGED TO H SPECIFICATION).
THEY OCCUR ON LINE -
 18.
```

```
S T Y L E   VERSION 1.0 AUG 72

THERE WERE ERRORS DETECTED THIS RUN.

THERE WERE ANSI VIOLATIONS DETECTED THIS RUN.
```

APPENDIX B

USE AT UNIVERSITY OF COLORADO

This section is intended as an aid to users of STYLE at the University of Colorado. The STYLE system is available through the Department of Computer Science library. It is a relocatable binary disk file named STYLE. It is available through either the batch or the time-sharing environments as follows:

Batch:
```
job card
account card
GET,STYLE/UN=X561.
STYLE.
```

Time-Sharing:
```
EXECUTE,OLD,STYLE/UN=X561.
RNH,MA=33000.
```

As a matter of convenience, a standard driver is available with STYLE. Input to the driver sets up the KEYS, files, and field length necessary for a particular run. (The user, however, must require enough field length to load.) The driver is dependent on CDC 6000 KRONOS operating system conventions. (It is written in COMPASS). Input to the driver is assumed to be on the file INPUT as follows:

P_1 = value, P_2 = value, ..., P_{n-1} = value, P_n = value.

where P_i may be:

K1 - KEYS(1), right most column for text
K2 - KEYS(2), left margin for text of comments
K3 - KEYS(3), right most column for labels
K4 - KEYS(4), label increment
K5 - KEYS(5), indentation for continuations
K6 - KEYS(6), sequencing increment
K7 - KEYS(7), indentation for DO-loops
K8 - KEYS(8), blank comments
K9 - KEYS(9), asterisk comments
K0 - KEYS(10), table of labels
SP - KEYS(11), Hollerith break character
N - max. # continuation cards on input
ID - 3 character sequence identifier
I - input file
L - output file
P - punch file file
F - reposition FORMATs

If a parameter is not specified, its default value is set. If a parameter is not equivalenced, its alternate value is set. The parameters are order independent; however, SP, N, ID, I, L, P, and F must appear on a separate card.

The defaults and alternates appear below:

Parameters	Default Value	Alternate Value
K1	64	None
K2	3	None
K3	5	None
K4	10	None
K5	1	None
K6	10	None
K7	2	None
K8	0	1
K9	0	1
K0	0	1
SP	*	1
N	2	0
ID	STL	None
I	INPUT	TAPE5
L	OUTPUT	TAPE6
P	0 (no punch)	PUNCH
F	1 (reposition)	0 (do not reposition)

K8, K9, K0, and F are never equivalenced. They behave as on-off switches for their appropriate functions. The first 7 KEYS have no alternates.

Using the example of Appendix A, the following sets of inputs request identical functions.

Sequence 1:

```
K1=50, K2=3, K3=3, K4=5, K5=2, K6=10, K7=3, K8=0, K9=0, K0=1
SP=*, N=0, ID=TST, I=INPUT, L=OUTPUT, P=PUNCH, F.
```

Sequence 2:

```
K1=50, K3=3, K4=5, K5=2, K7=3, K0.
N, ID=TST, P, F.
```

Sequence 3:

```
K1=50, K7=3, K4=5, K3=3, K5=2, KO.
ID=TST, N, P, F.
```

APPENDIX C:

INSTANT REFERENCE MANUAL

TABLE 1

User Controlled Options

IDENT

3 word array containing a 3 letter identifier, one letter per word, left justified - to be inserted in columns 73-75

N

maximum number of continuation cards

IDEV

5 word array defining the device numbers having the following meaning:

IDEV(1) - input file
IDEV(2) - output file
IDEV(3) - punch file (0 for no punched output)
IDEV(4) - intermediate file
IDEV(5) - intermediate file for formats (0 to leave formats where appear)

KEYS

11 word array defining the following options:

	default
KEYS(1) - rightmost column edited text may appear	64
KEYS(2) - left margin for text in comment statement (negative leave as is)	3
KEYS(3) - column to right justify edited statement label	5
KEYS(4) - increment value (plus beginning value) for edited statement labels	10
KEYS(5) - indentation for text on a continuation	1
KEYS(6) - increment value (plus beginning value) for sequencing in columns 77-80	10
KEYS(7) - indentation for text within DO-loops	2
KEYS(8) - blank comment statements 0 - remove 1 - leave	remove
KEYS(9) - comment statement containing only asterisks 0 - remove 1 - leave	remove
KEYS(10) - table of old vs. new statements labels 0 - do not print 1 - print	no table
KEYS(11) - optional break character to delimit Hollerith constants, left justified in words. It will be changed to an H specification. (0 - no character)	

TABLE 2

Fixed Editing Features

1. continuation cards are indicated by a star (*) in column 6

2. each DO-loop ends on its own CONTINUE card

3. if indentation proceeds to such a state that only 10 spaces remain, all further indentation ceases until the indentation backs out.

4. FORMATs have a unique labelling sequence:
   starting value 99999/10**(6-KEYS(3))
   decrement by one passing up any values of KEYS(4)

5. equal sign (=) - one space to either side (one before, one after)

6. operators +, -, .OR., .AND., - one space to either side

7. operators *, /, relationals (.GT., .GE., .EQ., .NE., .LE., .LT.)
   **, - no space to either side

8. unary operators (-, .NOT.) treated as a single unit along with their number or variable

9. commas (,) - one space to right only (excepting complex constants)

10. left parenthesis as delimiter - no space to right

11. right parenthesis as delimiter - no space to left

12. no space between nested parentheses

13. slash (/) in DATA and block COMMON statements - treated as left and right parenthesis delimiter (comment 10 and 11)

14. subscripts - no spacing within any subscript

15. Hollerith constants:
    FORMATs - split if necessary to fit column restraints set up by KEYS(1)
    DATA and CALL - no constant may be larger than column restraints the user himself has set (if so, unexpected results occur). These cannot be split.

16. normal English spacing conventions are followed

17. sequencing restricted to columns 77-80

18. user defined identifier restricted to columns 73-75

19. column 76 varies from A-Z, 0-9 providing 36 unique labels for separate subprograms

20. unreferenced labels are deleted

21. unreferenced FORMATs are flagged

TABLE 3

Acceptable Non-ANSI FORTRAN

1. statements
    1. BUFFER IN
    2. BUFFER OUT
    3. DECODE
    4. ENCODE
    5. ENTRY
    6. IDENT
    7. IMPLICIT
    8. INPUT
    9. NAMELIST
    10. OUTPUT
    11. OVERLAY
    12. PRINT
    13. PROGRAM
    14. PUNCH
    15. SECTION
    16. SEGMENT
    17. SEGZERO

2. character data specifications
    1. Z - hexadecimal
    2. Ø - octal
    3. R - right justified, zero fill
    4. L - left justified, zero fill
    5. T - tab function

3. special character delimiter for strings - changed to Hollerith constant

4. subscripted subscripts; any invalid subscript expression

5. Hollerith constant in function definition

TABLE 4

Errors - IERR(2)

ERROR CODE	MEANING
1.	STATEMENT OUT OF ORDER
2.	END LINE MISSING
3.	STATEMENT ILLEGAL IN BLOCK DATA
4.	NULL PROGRAM
5.	STATEMENT REQUIRES LABEL
6.	END NOT PRECEDED BY TRANSFER OF CONTROL
7.	NO EXECUTABLE STATEMENT IN SUBPROGRAM
8.	UNRECOGNIZABLE STATEMENT
9.	STATEMENT NUMBER CONTAINS NON-NUMERIC CHARACTER
10.	VARIABLE LENGTH TOO LONG MAXIMUM IS SIX
11.	DUPLICATE STATEMENT NUMBER
12.	IMPROPERLY NESTED PARENTHESES
13.	PARENTHESES NESTED TOO DEEPLY
14.	DO LOOP PARAMETER ERROR: IMPROPER INDEX PROBABLY A CONSTANT
15.	IMPROPER OR MISSING STATEMENT NUMBER IN DO
16.	DO LOOP PARAMETER SHOULD BE INTEGER VARIABLE OR CONSTANT
17.	SYNTAX ERROR
18.	SYNTAX ERROR: EXPECTING END OF STATEMENT
19.	STATEMENT EXCEEDED MAXIMUM OF 19 CONTINUATION CARDS
20.	SYNTAX ERROR: PUNCTUATION PROBABLY MISSING =
21.	SYNTAX ERROR: PUNCTUATION PROBABLY MISSING (
22.	SYNTAX ERROR: PUNCTUATION PROBABLY MISSING,
23.	SYNTAX ERROR: STATEMENT NUMBER IS REQUIRED ON CONTINUE STATEMENT
24.	SYNTAX ERROR: NON-NUMERIC CHARACTER IN DATUM
25.	SYNTAX ERROR: IMPROPER PUNCTUATION
26.	SYNTAX ERROR: PUNCTUATION EXPECTED , OR END OF STATEMENT
27.	SYNTAX ERROR: PUNCTUATION PROBABLY MISSING (OR EXTRA) OR / AFTER)
28.	SYNTAX ERROR: PUNCTUATION PROBABLY MISSING) OR EXTRA (
29.	SYNTAX ERROR: PUNCTUATION PROBABLY MISSING) OR /

ERROR CODE	MEANING
30.	SYNTAX ERROR: ARITHMETIC EXPRESSION PROBABLY VARIABLE OR OPERATOR OUT OF ORDER
31.	STATEMENT NUMBER HAS MORE THAN FIVE DIGITS
32.	OCTAL NUMBER HAS MORE THAN FIVE DIGITS (PAUSE OR STOP)
33.	NUMBER CONTAINS NON-OCTAL DIGIT
34.	SYNTAX ERROR: NON-NUMERIC CHARACTER IN EXPONENT OF DECIMAL CONSTANT
35.	SYNTAX ERROR: ILLEGAL CHARACTER IN FIELD
36.	SYNTAX ERROR: IN EXPRESSION
37.	SYNTAX ERROR: INCOMPLETE LIST, ENDED BY , OR EMPTY
38.	VARIABLE MUST BEGIN WITH ALPHA
39.	SYNTAX ERROR: MISSING -TO- IN ASSIGN
40.	SYNTAX ERROR: MISSING OR IMPROPER VARIABLE IN ASSIGN
41.	ILLEGAL STATEMENT IN LOGICAL IF
42.	SYNTAX ERROR: MISSING VARIABLE OR EXTRA , IN LIST
43.	SYNTAX ERROR: PUNCTUATION EXPECTED , OR EOS AFTER /
44.	PROBABLY ERROR IN FORMAT
45.	MISSING LABEL ON FORMAT
46.	TOO MANY CONTINUATION CARDS WITH SPACE
47.	UNDEFINED LABEL IN STATEMENT
48.	FORMAT UNREFERENCED
49.	UNDEFINED LABEL IN TEXT
50.	COULD NOT FIT SPACING RESTRAINTS
99.	PREMATURE END OF STATEMENT, SYNTAX INCOMPLETE

TABLE 5

System Errors

ERROR	MESSAGE	ACTION
STOP 1 -	BUFFER OVERFLOW, CHECK MAX. NO. OF CONTINUATIONS.	Respecify N rerun
STOP 2 -	OVERFLOW OF LABELS, INCREASE DIMENSION OF LDEF AND LREF.	Increase dimension LDEF, LREF change max recompile
STOP 3 -	DO STATEMENTS NESTED TOO DEEPLY, INCREASE FIRST DIMENSION OF LAB.	Increase dimension LAB Change max recompile
STOP 4 -	ARRAY OVERFLOW, INCREASE DIMENSION OF LENGTH,	Increase dimension length change LEX recompile
STOP 5 -	N (NUMBER OF CONTINUATION CARDS) INCORRECTLY SPECIFIED. MUST BE NON-NEGATIVE INTEGER LESS THAN 19.	Respecify N rerun
STOP 6 -	HOLLERITH CONSTANT IN DATA OR CALL TOO LONG FOR SPECIFIED COLUMN BOUNDARY i Where i is the boundary specified by KEYS(1).	Respecify KEYS(1) rerun