LBID-086/1 c.1

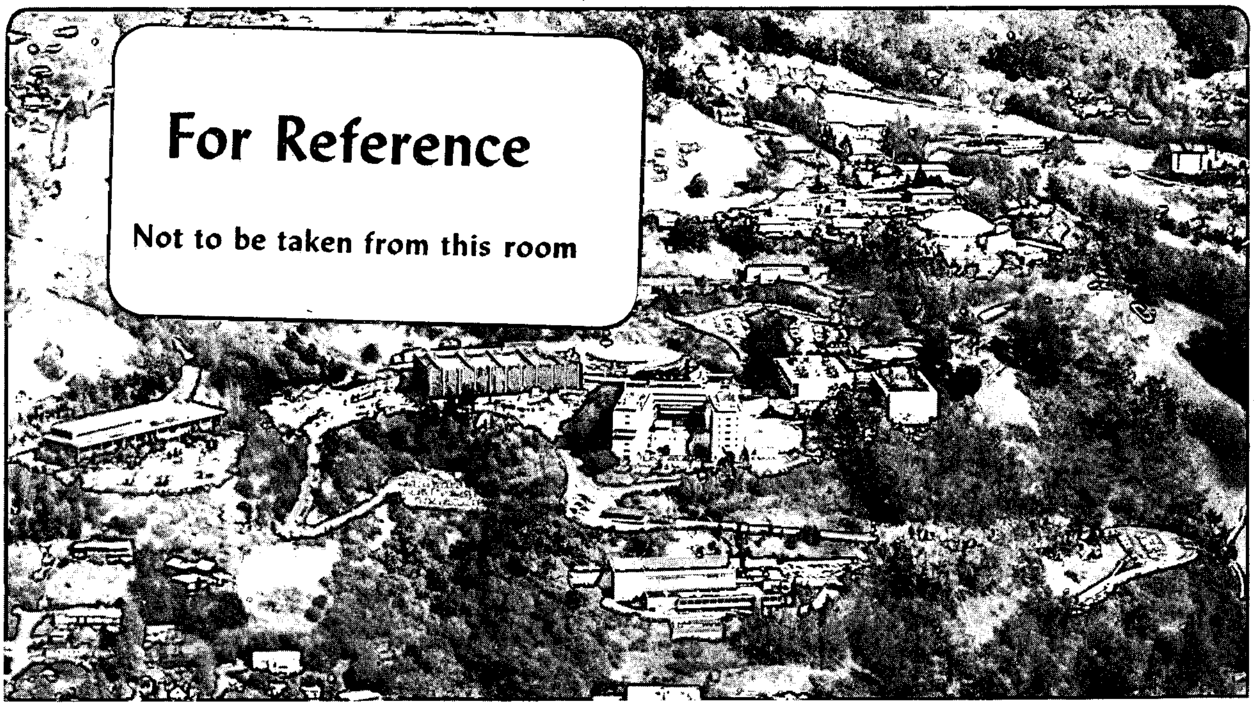
Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA, BERKELEY

Engineering & Technical
Services Division

BKY PROGRAMMING SYSTEMS BULLETIN #1

R. Friedman and Control Data Corp.

August 1979



Prepared for the U. S. Department of Energy
under Contract W-7405-ENG-48

32

Lawrence Berkeley Laboratory Computer Center
Systems Programming Group
University of California
Berkeley, CA. 94720

<u>BKY Programming Systems Bulletin #1</u>

1st Edition (August 79)

<u>Guidelines for Converting FTN4 Programs to FTN5 and the New FORTRAN-77 Standard</u>

<u>R.Friedman</u> (<u>LBL</u>),
and CONTROL DATA CORP.

CONTROL DATA will be releasing their FORTRAN 77 compiler, <u>FTN5</u>, by September, 1979. It is anticipated that FTN5 will be made available on the BKY 7600 system by mid-1980.

This Bulletin is based on technical material prepared by CONTROL DATA detailing those FORTRAN language features that are changing as a result of the new FORTRAN-77 standard and their implementation of the FTN5 compiler. Suggestions to minimize conversion problems are offered here, as well as strategies to make programs FTN4/FTN5 compatible. Actions taken by the FTN4 to FTN5 conversion program <u>F45</u> are described in each instance.

INTRODUCTION:

The major change between FTN4 and FTN5 is the implementation of the new ANSI FORTRAN 77 standard. The new standard provides a variety of extensions to the existing FORTRAN language. Some of these -- IF-THEN-ELSE structures, a PARAMETER statement, a CHARACTER data and variable type -- are completely new to FORTRAN, but should be familiar constructs to users of other high level languages. Features in this category should not affect the workability of existing FTN4 programs, but should add considerably to readability and ease of coding for programs developed under FTN5.

A few features in FORTRAN change functionally with the new standard, though not syntactically. These features are especially worth noting, since FTN5 will compile them without error, though program actions may be altered. These include the computed GOTO and the DO-loop:

GOTO(...),IV	Under FTN5, the computed GOTO will "fall through" to the next statement if the control variable is smaller or larger than the number of statement labels specified in the GOTO statement. With FTN4, this situation causes a fatal error diagnostic abort.
DO19J=J0,JF	DO-loops will execute a minimum of 0 times, rather than 1 time. Thus, if J0 is greater than JF in this example, the loop will be skipped over. With FTN4, the loop would be executed once, with J=J0. An FTN5 control card option will be available to defeat this.

In addition to these changes, some obsolete or little used features will be eliminated and additional special features will be added. New and more powerful compiler control directives will be available, including directives which allow conditional compilation of sections of source code. The C$DEBUG package will be discontinued, though its array checking capability will be available through a global parameter on the FTN5 control card.

This bulletin details the FTN4 features that will be modified or discontinued under FTN5. Alternative coding procedures are suggested that may be used to minimize or eliminate the need for conversion. Where possible, coding practices common to both FTN4 and FTN5 are suggested. Programmers are encouraged to use these simple equivalents.

A FORTRAN Conversion Program, F45, will be made available soon (Fall '79 on BKY 7600). It will handle a substantial portion of the required changes automatically as indicated below, and will flag for manual changing those conversions that may have side-effects or require reprogramming. Use of F45 is described in a subsequent Programming Systems Bulletin, available through the LBL Computer Center Library.

Exponentiation:

FTN5 will evaluate successive exponentiation right to left. FTN4 performs such evaluations from left to right.

E.g.: A**B**C = A**(B**C) with FTN5, = (A**B)**C with FTN4.

===>>: Use parentheses to explicitly specify the desired order of evaluation. In the example, write A**B**C explicitly as (A**B)**C to get the same result as FTN4.

Abbreviations:

FTN5 will not allow the abbreviations for logical operators and truth values { .A., .O., .N., .X., .T., .F. } which are presently allowed in FTN4.

===>>: Use full keywords { .AND., .OR., .NOT., .XOR., .TRUE., .FALSE. } F45 will expand all such abbreviations.

Numbered Common Blocks:

FTN5 will not allow numbered common blocks. F45 conversion program will rename numbered common blocks with unique names of the form Znnnnn, where nnnnn is derived from the common block number taken to the base 36, using the digit 0, letters A thru Z, and the digits 1 thru 9.

===>>: Use symbolic names for all labeled common blocks.

END Statements:

FTN5 will consider ommision of an END statement to be a fatal error. Continuation of END statements will not be allowed.

===>>: Include an END statement on a single line in each program unit.

Array Reference Subscripts:

FTN4 allows an unsubscripted array reference to designate the first element of the array in contexts where such a reference cannot be interpreted as a reference to the entire array, as in:

```
DIMENSION A(5)
B=A
```

is considered equivalent to:

```
DIMENSION A(5)
B=A(1)
```

FTN4 also allows trailing subscript values to be omitted and they are assumed to be one. For example:

```
DIMENSION A(10,10,10)
B=A(4,3)
```

is considered equivalent to:

```
B=A(4,3,1)
```

FTN5 will consider such references to be fatal errors. Unsubscripted array references will be allowed only in contexts in which the whole array is referred to: parameter lists, common, data statements, etc.

===>>: Specify all subscripts explicitly whenever such a reference is intended. F45 will supply the implied subscripts to any array name in contexts not referring to the entire array.

<u>Continuation</u> <u>Lines</u>:

FTN5 requires that continuation lines have blanks in columns 1-5, though FTN4 presently allows other characters to appear in those positions.

===>>: Begin all continuation lines with blanks in columns 1-5. F45 will blank these columns.

In FTN5, a blank line will be considered a comment, but will not break a continuation sequence. FTN4 considers such a line to break continuation sequences.

===>>: Avoid blank lines with respect to continuation sequences. F45 converts a continuation line preceded by a blank line into an initial line.

Comment Lines:

FTN5 will not recognize a comment line denoted by the character $ in column 1. Only C or * are valid.

===>>: Use the character C or * in column 1 to denote comment lines. F45 will convert $ to either * or C depending on the F45 control statement CC parameter.

Multiple Statements Per Line:

FTN5 will not permit multiple statements to appear on a single line. FTN4 allows this construct, using the character $ as a statement separator, as in:

```
A = B  $  C = D
```

===>>: Avoid multiple statements on a line. Write each statement on a separate line:

```
A = B
C = D
```

F45 will convert multiple statements into separate lines.

IF Statements:

FTN5 will not permit 2-branch IF statements, either logical or arithmetic.

===>>: Replace 2-branch arithmetic IF's with 3-branch IF's:

```
          IF (e) s1, s2
      becomes:
          IF (e) s1,s2,s1
```

Replace 2-branch logical IF's with a 2-statement IF-GOTO sequence:

```
          IF (e) s1,s2
      becomes:
          IF (e) GO TO s1
          GO TO s2
```

F45 will do these conversions automatically.

Octal Constants:

FTN5 will require that octal constants take the form

```
O"nnnn"
```

rather than the

```
nnnnB
```

form presently allowed by FTN4.

In addition, FTN5 interprets an octal constant as an unsigned, typeless operand. A unary plus or minus preceding an octal constant is prohibited in a DATA statement in FTN5. If used in an executable statement it has the effect of causing an implicit type conversion from octal to integer which may result in other unexpected type conversions.

The FTN4 statements:

```
DATA M /23B/
DATA N /-333B/
I = -30B
J = 27B + 15B
```

would require conversion to:

```
DATA M /O"23"/
DATA N /O"77777777777777777444"/
I = .NOT. O"30"
J = O"27" + O"15"
```

Note that the second DATA statement uses the 60-bit one's complement value for -333B in order to avoid the use of an operator in a DATA statement.

The third assignment statement uses the logical operator for one's complement, .NOT., which would not be allowed in a DATA statement. Likewise, the statement could have been written:

```
I = O"77777777777777777747"
```

The final assignment statement uses an operator, since a unary operation is not involved.

===>>: Programs which use octal constants will require conversion. F45 will convert octal values as illustrated. To avoid the necessity for conversion, use decimal equivalents of octal values wherever possible:

```
DATA M /19/
DATA N /-219/
I = -24
J = 23 + 13
```

Signed Typeless Constants and Operands:

In an assignment statement or statement function definition, F45 will interpret a unary plus or minus operator followed by a typeless operand as an unsigned constant prefixed by an operator, and converts the constant as indicated by the context. The statements:

```
WORD1 = +222B
WORD2 = -3HXYZ
WORD3 = -333B
WORD4 = -SHIFT(STRING,24)
```

are translated to:

```
WORD1 = O"222"
WORD2 = (.NOT.3HXYZ)
WORD3 = (.NOT.O"333")
WORD4 = (.NOT.SHIFT(STRING,24))
```

F45 replaces the unary minus with .NOT. and encloses the negated operand with parentheses. F45 deletes unary plus.

Hollerith Constants:

FTN5 will not permit Hollerith constants that exceed 10 characters in length. (A new data type, CHARACTER, is provided to handle longer strings.) In addition, Hollerith constants will be treated as unsigned, typeless operands, prohibiting the use of a unary plus or minus with such constants. (This restriction is similar to that described above for octal constants.) Thus,

```
I = -3HABC
```

should be rewritten as:

```
I = .NOT. 3HABC
```

The four forms of Hollerith constants:

```
             "......"
             nH.....
             nR.....
             nL.....
```

will continue to be allowed in FTN5, subject to the restriction that n ≤ 10

===>>: Restrict Hollerith constants to a maximum of 10 characters in length. Constructs of the form:

```
                  DIMENSION I(2)
                  DATA I /16H THIS IS TOO LONG /
                     .
                     .
                  CALL SUB(J,K,"THIS IS EVEN LONGER")
                     .
```

should be rewritten as:

```
                  DIMENSION I(2),L(2)
                  DATA I /10HTHIS IS TO, 6HO LONG /
                     .
                  L(1)="THIS IS EV"
                  L(2)="EN LONGER"
                  CALL SUB(J,K,L)
                     .
```

F45 will handle these modifications. Hollerith constants can appear in three contexts: in an expression, in a DATA statement, or as an actual argument in a subprogram call. F45 treats these three cases differently.

In an expression, FTN4 ignores all characters after the leftmost ten characters. F45 truncates to ten characters any long Hollerith constant in an expression. The resulting constant produces the same result under FTN5 as the original under FTN4.

In DATA statements, FTN4 allows a long constant to initialize successive elements of an array. If a long Hollerith constant is the last item in a data list, F45 breaks the long constant into a series of ten-character constants, possibly followed by one shorter constant. If the long constant is not the last constant in the data list, F45 truncates the long constant to ten characters. The user should check that the code generated by F45 produces the same results as the original code, especially if the program depends on trailing zeroes.

When used as an actual argument, a long Hollerith constant is not shortened. For the Hollerith forms nH or "...", F45 converts the constant to a type CHARACTER constant with the form '...' . For a long constant of the form nR or nL, F45 replaces the constant with an integer array name that it creates. The array name is Znnnnn, where nnnnn is a unique combination of digits and letters.

DATA Statements:

FTN5 will not permit the non-ANSI "alternate" DATA statement which is presently accepted by FTN4:

```
DATA (I=5), (J,K=6,7)
DATA (X=3.25)
```

===>>: Use the standard form of the DATA statement:

```
DATA I /5/, J,K /6,7/
DATA X /3.25/
```

FTN4 presently allows 3 non-standard forms of the constant list in a DATA statement:

```
(constant list)
rf*(constant list)
rf(constant list)
```

where rf represents a repetition count.

The first two forms result in syntactic ambiguity, since constructs such as:

```
(1.2,3.4)
10*(5.6,7.8)
```

may be interpreted as containing either a single COMPLEX constant or a list of two REAL constants. Accordingly for FTN5, the form rf*(constant list) specifies a repeated series of complex constants, while the form rf(constant list) specifies a repeated series of real constants.

Also, FTN5 will not allow signed octal or Hollerith constants in DATA statements, as described earlier.

===>>: Eliminate all redundant parentheses from DATA statements. Omit the * when using a repetition factor with a constant list. Thus,

```
REAL A,B,C,D,E,F
DATA A,B /(1.2,3.4)/
DATA C,D,E,F, /2*(5.6,7.8)/
```

should be rewritten as:

```
                    DATA A,B /1.2,3.4/
                    DATA C,D,E,F, /2(5.6,7.8)/
```

Selection of the appropriate forms should be sufficient to avoid conversion except in the case where a single COMPLEX constant is used with a repetition factor in FTN4:

```
                    DATA A,B /2*((3.4,5.6))/
```

The double parenthesis required to indicate the presence of a COMPLEX constant rather than a list of REAL constants will generate an error in FTN5. Avoid this construct by listing COMPLEX values in full:

```
                    DATA A,B /(3.4,5.6),(3.4,5.6)/
```

F45 removes redundant parentheses automatically, and converts the forms:

```
               rf*(real,real)
               rf*((real,real))
               rf*(const1,const2,const3)
```

to the forms:

```
               rf(real,real)
               rf*(real,real)
               rf(const1,const2,const3)
```

COMPLEX Numbers:

FTN4 allows the use of COMPLEX expressions in arithmetic IF statements, whereas FTN5 does not. FTN4 permits either or both operands in a relational expression to be of type COMPLEX, though when the operator is other than .EQ. or .NE. it compares only the REAL parts of the operands. FTN5 allows relational expressions containing COMPLEX operands only when the operator is .EQ. or .NE.

===>>: Use the REAL function to convert COMPLEX numbers to type REAL when they appear in arithmetic IF statements or in relational expressions with operators other than .EQ. or .NE.

```
               COMPLEX C,CC
                    .
                    .
               IF (C) 10,20,30
               IF (C.GT.CC) GOTO 100
```

should be re-coded as:

```
IF (REAL(C)) 10,20,30
IF (REAL(C).GT.REAL(CC)) GOTO 100
```

F45 will convert these expressions automatically.

STOP and PAUSE Statements:

FTN4 allows a STOP or PAUSE statement to include a quote-delimited Hollerith string. (The string is sent to the job's DAYFILE.) FTN5 will require that such Holleriths be converted to apostrophe-delimited character constants (strings). Thus,

```
STOP "THATS ALL FOLKS"
```

becomes:

```
STOP 'THATS ALL FOLKS'
```

===>>: Since the apostrophe (single-quote) delimited character type does not exist in FTN4, programs using quote delimited strings on STOP or PAUSE statements are incompatible. F45 will convert these statements automatically.

TYPE Statements:

FTN5 will not allow the optional keyword TYPE presently accepted by FTN4, as in:

```
TYPE INTEGER A,B
TYPE REAL I,J,K
```

FTN5 will also not accept the abbreviation DOUBLE for DOUBLE PRECISION, as in:

```
     DOUBLE DD
or   TYPE DOUBLE DD
```

And, FTN5 will not accept the declaratives:

```
     ECS
or   TYPE ECS
```

===>>: Omit the word TYPE from any such statements. Use the full keyword DOUBLE PRECISION. Use LEVEL 3 statements to indicate ECS memory storage. These will be converted automatically by F45.

LEVEL Statement:

FTN4 allows a LEVEL statement, which contains names of variables, arrays, and dummy arguments, though variables and arrays so used must be contained in COMMON blocks.

FTN5 will require COMMON block names to be listed in LEVEL statements, rather than array or variable names. Dummy arguments will continue to be accepted. Thus:

```
COMMON /LCMB/ X,Y,Z
LEVEL 2, X,Y,Z
```

becomes:

```
COMMON /LCMB/X,Y,Z
LEVEL 2,/LCMB/
```

===>>: Use F45 to convert LEVEL statements automatically. FTN4 programs with LEVEL statements will be incompatible with FTN5.

ENTRY Statements:

FTN5 permits a dummy argument list with each ENTRY statement and will require that an external reference to an ENTRY use an actual argument list that agrees in order, number and type with the dummy list.

FTN4 does not allow dummy argument lists other than in a header statement and uses the list specified in the header as the dummy list for all ENTRY points in the program unit.

===>>: Routines using ENTRY statements will be incompatible and require conversion. F45 will add a dummy argument list identical to that specified in the header to each ENTRY statement. Note that program units containing ENTRY statements will appear to compile correctly under FTN5 without conversion, though run-time errors will result.

SLITE and SLITET:

FTN5 will not support the sense light facility CALL SLITE and CALL SLITET. This feature is considered archaic and its original purpose of program-to-console-operator communication has never been supported.

===>>: Use logical variables instead, or write your own FORTRAN subroutines to perform a similar function. F45 will flag such useages as requiring manual conversion.

READEC and WRITEC:

FTN5 will not support the library utility subprograms READEC and WRITEC.

===>>: Use the preferred MOVLEV library utility, as in:

```
CALL READEC(A,B,10)  becomes  CALL MOVLEV(B,A,10)
CALL WRITEC(C,D,15)  becomes  CALL MOVLEV(C,D,15)
```

F45 will handle this translation automatically. However, users of READEC and WRITEC should be aware that a serious problem will develop if these routines are being used via an indirect SCM variable pointer into the LCM array. F45 will not detect this useage. E.g., WRITEC permits

```
LCMP = 40001B
CALL WRITEC(SORES(1),LCMP,1000)
```

where LCMP and SORES are both SCM resident. LCMP is recognized by WRITEC to be SCM resident, and its value, in this case 40001, is taken as the destination address in LCM to be used to transfer 1000 words from SORES(1). Programmers must inspect their use of READEC and WRITEC to avoid this situation. Reprogramming may be necessary (or, possibly, READEC and WRITEC may be made available for FTN5 users).

Computed GOTO:

Under FTN4, execution of a computed GOTO statement results in a run-time error when the value of the index variable is less than or equal to 0, or is greater than the number of labels specified in the statement. FTN5 specifies that a computed GOTO "falls through" and continues execution in such a case.

The code shown will generate an error in FTN4, but will proceed to statement 5 under FTN5:

```
      J = 5
      ...
      GOTO (10,20,30),J
 5    X = Y
 10   X = X * X
      ... etc
```

===>>: Programs coded to insure that computed GOTO indices are within range of the GOTO statement will not require conversion. However, programs which rely on FTN4 run-time diagnostics to detect out-of-range indices should be recoded to handle these errors, as in:

```
      GOTO(10,20,30),J
      PRINT 50
   50 FORMAT (" ERROR IN COMPUTED GOTO " )
      CALL ABORT
   5  X = Y
   10 X = X * Y   etc.
```

Inclusion of such additional code will cause non-fatal informative diagnostics under FTN4 since FTN4 will assume that the lines following the GOTO cannot be reached during execution.

F45 conversion program will add the statement CALL GOTOERR under each computed GOTO statement to make the resultant processing identical to FTN4.

DO-Loops:

FTN4 executes the range of a DO-loop at least 1 time. FTN5 will skip execution of the loop if the iteration count is initially 0. An option on the FTN5 control card will allow the user to specify a minimum 1-time execution of DO loops if desired.

===>>: Specifying DO=OT on the FTN5 control card will compile programs that are guaranteed to behave identically under FTN4 as far as one-trip DO loops are concerned. A more effective procedure would be to design code in such a way that minimum loop behavior is explicitly specified, as in:

```
      DO 10 I = J,K
        ...
   10 CONTINUE
```

can be written for 0-trip execution:

```
      IF (J.GT.K) GOTO 20
      DO 10 I=J,K
        ... etc
   10 CONTINUE
   20  ... etc
```

F45 will take no action which will affect the minimum loop execution. Programs which will be affected by the change to FTN5 will require use of the DO=OT option at compile time.

Alternate RETURNS:

The syntax of the alternate RETURNS feature in FTN4 will be modified considerably under FTN5. The keyword RETURNS will be eliminated and the statement labels for alternate return points will be included in the parameter list in both CALL statements and SUBROUTINE header statements. Such labels will be preceded by a * to distinguish them from items in the parameter list.

In addition, the statement

```
RETURN v
```

where v indicates the value of the dummy parameter which designates the return point, will be replaced by

```
RETURN i
```

where i is an integer variable referencing the i-th item in the alternate returns list. Thus, the FTN4 code:

```
CALL SUB(X,Y) , RETURNS(10,20,30)
      ...
SUBROUTINE SUB(A,B) , RETURNS(I,J,K)
      ...
RETURN I
      ...
RETURN J
      ...
RETURN K
      ...
```

would have as its FTN5 equivalent: (as converted by F45)

```
CALL SUB(X,Y,*10,*20,*30)
      ...
SUBROUTINE SUB(A,B,*,*,*)
      ...
RETURN 1
      ...
RETURN 2
      ...
RETURN 3
      ...  etc.
```

===>>: Conversion will be necessary for any program using the alternative RETURNS feature. Users desiring to avoid conversion completely may consider using some other method of effecting the desired control flow. For example, the code below produces the same flow control as the example above, but is acceptable to both FTN4 and FTN5:

```
      CALL SUB(X,Y,I)
      GOTO (10,20,30),I
          ...
      SUBROUTINE SUB(A,B,I)
          ...
      I=1
      RETURN
          ...
      I=2
      RETURN
          ...
      I=3
      RETURN
          ...  etc.
```

Hollerith *....* in FORMAT:

FTN5 will not accept asterisk-delimited Hollerith strings in FORMAT statements presently allowed by FTN4. A new apostrophe-delimited string will be provided. Also, the FTN4 accepted quote-delimited string will be retained.

===>>: Use the quote delimited form of Hollerith strings in FORMAT statements to avoid the need for conversion. Thus:

```
FORMAT(* A MESSAGE *) becomes FORMAT(' A MESSAGE ')
```

F45 will change both quote and asterisk forms to the new single-quote (apostrophe) form.

X FORMAT Descriptor:

FTN5 will require that the X FORMAT descriptor be prefixed always with a non-zero, unsigned integer constant.

===>>: Avoid use of the form 0X, and replace X with 1X wherever it occurs. Use the T edit descriptor in place of the -nX form. F45 will handle these changes, but will replace occurances of the -nX with TLn, where the TL descriptor is a new feature of the FORTRAN 77 standard which moves the indicated number of spaces left.

H FORMAT Descriptor on Input:

FTN5 will not permit the use of the H format descriptor on input operations.

===>>: Use arrays read and written in A format in place of H format. Thus,

```
      READ(1,30)
      PRINT 30
   30 FORMAT(20H                    )
```

can be replaced with

```
      INTEGER MESS(2)
      READ(1,30)MESS
      PRINT30,MESS
   30 FORMAT(2A10)
```

F45 will flag occurrences of this construct, but a manual change will be required by the user. The use of CHARACTER data is recommended in FTN5.

T0 FORMAT Descriptor:

FTN5 will not allow the T0 specification in FORMAT statements. FTN4 considers this equivalent to T1.

===>>: Use the T1 specification instead of T0. F45 will convert T0 to T1, but T0 in variable format cannot be detected.

Commas in FORMAT Statements:

FTN5 requires that commas be used to separate list items in a FORMAT statement after an X, nH, or quote-delimited edit descriptor. Such commas are optional under FTN4.

===>>: Include commas as separators in FORMAT statements. F45 will supply missing commas.

Variable FORMAT:

FTN5 will not permit a variable format specification to be in a simple non-character variable.

===>>: Convert simple variables to arrays if used as variable format:

```
          DATA  M /4H(I6) /
          READ (1,M) I
```

becomes:

```
          INTEGER M(1)
          DATA M(1) /4H(I6) /
          READ (1,M(1)) I
```

F45 will flag each appearance of a simple variable name used as a format specification but manual conversion will be required.

V and = Edit Descriptors:

FTN5 will not allow the V and = edit descriptors allowed in FTN4.

===>>: There is no direct equivalent of the V edit descriptor and its use should be avoided. The = edit descriptor is also without a direct FTN5 equivalent, though there are features for handling CHARACTER data output that should prove to be an effective substitute in some cases. F45 will flag each use of such a descriptor for manual conversion.

E Format Exponent Length:

FTN4 allows Ew.dDe where De indicates an explicitly specified exponent length. FTN5 disallows De but allows Ew.dEe.

===>>: Use E rather than D to specify the length of the exponent field. F45 will do the conversion automatically.

Double Precision Format Descriptors:

FTN5 will require that each Double Precision I/O list item correspond to exactly one repeatable format descriptor. FTN4 permits Double Precision items to correspond to two format descriptors.

The conversion program F45 will not recognize the use of two format descriptors (e.g. 2E20.10) for one Double Precision list item, so particular care should be taken to avoid such usage.

===>>: Use single D, E, F, or G format descriptors to output double precision variables. Note that equivalencing may be needed to print Double Precision variables in O format:

```
          DOUBLE PRECISION X           insert:
           ...                                   INTEGER A(2)

           ...                                   EQUIVALENCE(A,X)
          PRINT 20,X                change to:   PRINT 20,A
       20 FORMAT(1X,2O20)
```

I/O Lists:

FTN5 will not permit the use of redundant parentheses in input/output lists. FTN4 permitted redundant pairs of parentheses. Redundant parentheses can result in ambiguous statements. Consider:

```
      PRINT *, (2.31,8.)
```

According to the syntax, the I/O list could consist of either a single complex constant or two real constants. FTN4 will assume that such a list item is a complex constant. To avoid this ambiguity, FTN5 prohibits redundant parentheses, and will always treat (real,real) as a complex constant. F45 will remove redundant parentheses in I/O lists.

List Directed I/O:

FTN4 processes list-directed PRINT* and WRITE* statements differently. For PRINT*, a blank is output as the first character of each record and also as the first record of each line when a long record is continued on additional lines. For WRITE*, a blank is output only as the first character of each record. WRITE* also includes the quote symbols with character output.

FTN5 will process PRINT* and WRITE* identically, transmitting a blank for each line of output, but not including the delimiting symbols with character output.

===>>: Formatted output should be used to handle cases which will be affected by this change. F45 will not flag or translate such usage.

FTN4 List-directed READ* statements read enough data from the input line to satisfy the I/O list of the READ* statement. Extra data on the line not picked up by the READ* will be read by the next READ* statement. Under FTN5, list-directed READ* statements will always read data from a new line. Extra data not read from the line by one READ* is skipped by a subsequent READ*. This is a significant difference in operation between compilers, and programs that rely on this situation will have to be reprogrammed. F45 cannot detect this situation.

Hollerith Data in List-Directed I/O:

FTN4 assumes that an integer data item in a list-directed output list is a Hollerith constant if its absolute value exceeds 2**48-1. Such values are transmitted with an A10 specification. FTN5 will not provide this interpretation.

===>>: Use formatted output for printing Hollerith data. F45 will detect the direct use of Hollerith constants in list directed output statements and will translate such Holleriths to Character data. However, F45 cannot detect the use of variables which have been assigned (via DATA or var = nH... assignment statement) Hollerith values.

Output Statements:

FTN5 does not allow the following forms of output statements, as they are considered redundant and non-standard:

	use instead:
WRITE fmt	PRINT fmt
WRITE fmt,list	PRINT fmt,list
WRITE *,list	PRINT*,list
PRINT (unit,fmt) list	WRITE (unit,fmt) list
PRINT (unit,*) list	WRITE (unit,*) list
PUNCH (unit,fmt) list	WRITE (unit,fmt) list
PUNCH (unit,*) list	WRITE (unit,*) list

F45 will do these conversions.

READ Statements:

With FTN5 additional specifications are allowed on the READ statement. FTN4 READ statements will continue to behave identically under FTN5 without the addition of any of these specifications, unless an end-of-file is encountered during the execution of a READ. In this case, an error will be generated unless END=label has been specified, where label is a statement label contained in the same program unit as the READ statement, which indicates where control is to be transferred when an end-of-file is encountered.

Many FTN4 programs will not be affected by this change, though programs using the EOF or IOCHEC functions will require conversion. These functions will continue to be available under FTN5, but the END= specification must be added to the associated READ statements in order to avoid an error during input of an end-of-file.

===>>: F45 will add an END=label specification to all READ statements, where label is the label of the statement following the READ statement. If no such label exists, one will be generated.

```
statement:            will be converted to:
 READ(u) list            READ(u,END=label) list
 READ(u,fmt) list        READ(u,fmt,END=label) list
 READ fmt, list          READ(u,fmt,END=label) list
```

and conversion of other variations on the READ statement would be similarly made.

However, the following:

```
      READ (1,100) CAT
      IF (EOF(1).NE.0) GOTO 800
      ...
  800 ....
```

must be converted to:

```
      READ(1,100,END=800) CAT
```

Under FTN4, the IOCHEC function could be used to check for parity and other I/O errors in the previous I/O operation. FTN5 uses the specifiers IOSTAT and ERR for error processing. These specifiers are part of the READ statement itself:

```
READ(lfn,IOSTAT=var,ERR=n)
```

for example,

specifies that integer variable var is set by the READ operation to some positive value if a an error has occurred, and that processing continues at statement label n on error. If neither IOSTAT or ERR has been specified on a READ statement, and an error in reading occurs, FTN5 will terminate the program.

The IOCHEC function will still be available with FTN5 to check for parity errors.

Statement Functions:

FTN4 executes references to statement functions by replacing each dummy argument with the actual argument, without regard to type correspondence or side effects.

FTN5 will evaluate and type convert each argument before substitution. Side effects will be prohibited.

===>>: Avoid statement functions which depend on side effects or the avoidance of type conversion.

F45 will flag references to statement functions which contain a type mismatch but dependence on side effects cannot be detected. Manual conversion will be required.

Intrinsic Functions:

FTN4 allows the following names of intrinsic functions to be passed as actual arguments in subprogram calls:

```
AMAX0    DMAX1    MAX1
AMAX1    DMIN1    MIN0
AMIN0    FLOAT    MIN1
AMIN1    IDINT    REAL
AND      IFIX     SNGL
CMPLX    INT      OR
DBLE     MAX0     XOR
```

FTN5 disallows this.

===>>: Avoid passing the listed functions as actual arguments. F45 will flag such usage, but manual conversion will be required.

New Intrinsic Functions:

FTN5 provides the following new intrinsic functions:

```
ANINT    DPROD    LGE    LOG10
BOOL     EQV      LGT    MAX
CHAR     ICHAR    LLE    MIN
DDIM     IDNINT   LLT    NEQV
DINT     INDEX    LOG    NINT
DNINT    LEN
```

===>>: Avoid use of the names listed above in contexts which will cause them to be interpreted as references to intrinsic functions. F45 will add EXTERNAL statements wherever one of these names is used as a function to defeat FTN5's typing as intrinsic.

RANF:

FTN4 presently requires a dummy parameter for the RANF function. FTN5 will retain this function, but will require that it be called with no parameters:

```
    X = RANF()  instead of RANF(DUMMY)
```

===>>: F45 will convert these usages.

C$ DEBUG Package:

FTN5 will not provide the C$ DEBUG package available with FTN4. DEBUG directives should be removed from the source code of programs before compiling with FTN5. (FTN5 List directives use the C$ syntax and if not removed, C$ DEBUG directives will appear to be erroneous LIST directives).

FTN5 will provide a control card option to select checking of subscripts in array references.

===>>: F45 will delete statements beginning with C$. Also, some sort of post-mortem dump package has been advertised with FTN5 though its implementation on BKY may be delayed.

List Directives:

FTN5 will require minor changes to the syntax of the existing C/LIST directives for source program listing control:

FTN4	change to:
C/ LIST,ALL	C$ LIST(ALL)
C/ LIST,NONE	C$ LIST(ALL=0)

FTN5 will allow a variety of new LIST directives, as well as additional directives which can be used to specify conditional compilation of sections of source code, and several other features no available under FTN4. All these directives begin with a C$ in columns 1 and 2 and will be treated as comments if the appropriate FTN5 control card parameter is not specified.

===>>: F45 will make the appropriate conversion of C/LIST directives.

FURTHER INFORMATION:

The ultimate source of information on FORTRAN-77 is the standard itself, published by the American National Standards Institute (ANSI) as: ANSI X3.9-1978, Programming Language FORTRAN.

Information on CONTROL DATA's FTN5 compiler is found in their FTN5 Reference Manual, soon to be available.

Use of CDC's F45 FTN4 to FTN5 conversion program will soon be available as a BKY Programming Systems Bulletin.

All these items are or will be available from the Computer Center Library, Room 1245A/50B, Lawrence Berkeley Lab., University of California, Berkeley, CA. 94720 (415-486-5529).

Publication and/or availability of all documentation is announced in the BKY Computer Center Newsletter.

*