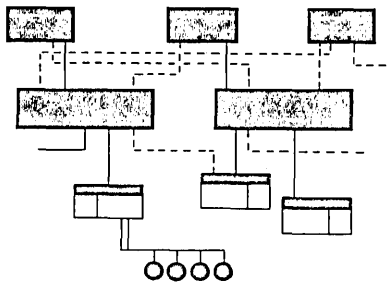
GIFT
General Internal
FORTRAN Translator

SYSTEM
SUPPORT
INFORMATION



ABSTRACT

This document describes the Compatibles/600 General Internal FORTRAN Translator (GIFT). Part I discusses the differences between FORTRAN II and the GIFT translation to FORTRAN IV, and significant features of GIFT. Part II discusses requirements for FORTRAN II programs to be translated and how to use GIFT, including deck setup and program options.

GENERAL ELECTRIC

GIFT

General Internal

FORTRAN Translator

September 1964

GENERAL ELECTRIC

COMPUTER DEPARTMENT

PREFACE

This manual describes the General Internal FORTRAN Translator (GIFT) for use with the Compatibles/600 computers. This program automatically translates FORTRAN II source programs into FORTRAN IV language. The level of presentation assumes reader familiarity with FORTRAN.

For additional information concerning FORTRAN IV for the Compatibles/600, refer to the GE-635 FORTRAN IV Reference Manual, CPB-1006.

GIFT is the General Electric Computer Department version of the SHARE Internal FORTRAN Translator (SIFT), described in the SHARE Internal FORTRAN Translator Users Manual, SHARE Distribution #1367 HS SIFT (PA), prepared by members of the SHARE FORTRAN Project, September, 1962.

Address any comments or questions about this publication to Technical Writing, General Electric Computer Department, Drawer 270, Phoenix, Arizona 85001.

CONTENTS

1. PROGRAM FEATURES

The General Internal FORTRAN Translator (GIFT) automatically translates a FORTRAN II source program into a FORTRAN IV source program. It eliminates certain incompatibilities between the two languages and modifies certain statements to conform with recommended or alternate formats. These changes brought about by the program are discussed in this chapter.

TRANSLATIONS EFFECTED BY GIFT

F Card

The F card, used to specify function or subroutine names which are arguments to other functions or subroutines, is replaced by an EXTERNAL Type statement. For example:

FORTRAN II	GIFT Translation
F SIN, COS, FUNC	EXTERNAL SIN, COS, FUNC

Function Names

- Elimination of Terminal F. Built-in, library, and arithmetic statement functions are no longer identified by a terminal F. GIFT therefore removes the terminal F from every function name. For example:

FORTRAN II	GIFT Translation
X = Y + SINF (A) *ADFUNF (Z**2)	X = Y + SIN (A) *ADFUN (Z**2)

- Fixed-Point Function Names. Fixed-point function names normally begin with I, J, K, L, M, or N rather than X. GIFT therefore manufactures INTEGER and REAL Type statements where appropriate. For example:

FORTRAN II	GIFT Translation
X = LASTF (Y)	REAL LAST X = LAST (Y)
I = XTRAF (1,K)	INTEGER XTRA I = XTRA (1,K)

why? Because the argument is real. To return an integer function, the fname should begin with X

- Changed Function Names. Many of the function names have been changed. GIFT replaces each of the FORTRAN II names listed below by the FORTRAN IV equivalent shown.

FORTRAN II Name	GIFT Translation	FORTRAN II Name	GIFT Translation
XABSF	IABS	INTF	AINT
XINTF	INT	MODF	AMOD
XMODF	MOD	XMAX1F	MAX1
XFIXF	IFIX	XSIGNF	ISIGN
XDIMF	IDIM	MAX0F	AMAX0
MAX1F	AMAX1	MIN0F	AMIN0
MIN1F	AMIN1	XMIN0F	MIN0
XMIN1F	MIN1	XMAX0F	MAX0
LOGF	ALOG		

- Other New Function Names. Many other new function names have been created. Every time GIFT encounters a variable whose name is the same as one of the new FORTRAN IV function names, it creates a name not used in the program being translated. It uniformly replaces all occurrences of the conflicting name with this created name, which is called an "insert variable." The insert variable is chosen so that its first letter determines the correct type for replacement.

  The list below includes the new FORTRAN IV function names recognized by GIFT:

SLITE	SLITET	SSWTCH	OVERFL
DVCHK	IABS	AINT	INT
AMOD	MOD	AMAX0	AMAX1
AMIN0	AMIN1	MIN0	MIN1
MAX0	MAX1	IFIX	ISIGN
IDIM	ALOG	CABS	STORE
CSIGN	PART	CEXP	CLOG
CSIN	CCOS	CSQRT	OR
AND	BOOL	COMPL	COM

However, names of subroutines and functions which conflict with new FORTRAN IV names cannot be merely replaced by insert variables. GIFT recognizes two degrees of severity of conflict:

1. Nonfatal example:

```
CALL ALOG (A,B)
CALL EXIT
END
```

   ALOG conflicts with the FORTRAN IV subroutine of the same name, but no usage of ALOG is constructed by the translator.

2. Fatal example:

```
CALL ALOG (A,B)
X = LOGF (A)
    becomes
CALL ALOG (A,B)
X = ALOG (A)
```

   This is a definite and fatal error. The program must be corrected by hand.

Boolean Statements

- Arithmetic, IF, and CALL Statements. Arithmetic, IF, and CALL statements with a B in column 1 are modified so that the Boolean operators +, *, and - are replaced by OR, AND, and COMPL functions, respectively. For example:

FORTRAN II	GIFT Translation
B A = C + D	A = OR (C,D)
B F = G* (-H)	F = AND (G, COMPL (H))
B IF (A*B) 10,20, 10	IF (BOOL (AND (A,B)))10, 20, 10

The call to BOOL is inserted to provide a zero-vs-nonzero test of all 36 bits of the result. In the FORTRAN II program, a transfer of control to the negative branch of the IF statement is impossible.

- Octal Constants. Each octal constant is replaced by an insert variable name; and a DATA statement is manufactured, specifying the variable and its corresponding value. Thus:

FORTRAN II	GIFT Translation
B P = R*77777	DATA Q000CT/O77777/ P = AND (R, Q000CT)

- Return Statements. Since the AND, OR, and COMPL functions obtain arguments from and return results to the algebraic rather than the logical accumulator, the Boolean RETURN statement is no longer meaningful. Thus, for example:

FORTRAN II	GIFT Translation
B RETURN	RETURN

Double-Precision and Complex Statements

- Alteration of Variable and Function Names. In FORTRAN IV, variable and function names rather than statements are specified as double-precision or complex. GIFT includes in an appropriate Type statement every variable and function name which appears in a statement containing a D or an I in column 1 and removes the D or I. For example:

FORTRAN II	GIFT Translation
I DIMENSION XI (2, 3), YI (5, 5, 5)	DIMENSION XI (2, 3), YI (5, 5, 5)
D ALPHA = BETA * GAMMA	DOUBLE PRECISION ALPHA, BETA, GAMMA
	COMPLEX XI, YI
	ALPHA = BETA * GAMMA

In FORTRAN IV, every function name in a double-precision or complex statement is prefixed by a D in double-precision statements and a C in complex statements. In addition, the terminal F is removed as

explained on page 1; and the name is included in a Type statement, as explained above. For example:

FORTRAN II	GIFT Translation
I Y = SINF (X)	COMPLEX Y, CSIN, X Y = CSIN (X)

- References to Double-Precision and Complex Variables. GIFT handles references to double-precision and complex variables in the following manner:

  1. In statements not preceded by D or I--If a reference to a double-precision or complex variable name appears in an arithmetic expression in a statement without a D or an I in column 1, that variable becomes an argument to the function SNGL (or function REAL) if it is unsubscripted or to the function PART if it is subscripted. The PART function also contains as arguments the name of the array and its length. At execution time, then, the desired part of the double-precision or complex pair is returned to the object program.

     If, however, the reference is an unsubscripted variable name used as an explicit argument to a function reference or a CALL statement, it is passed on unaltered. For example:

	FORTRAN II		GIFT Translation
I	DIMENSION A (5, 5)		DIMENSION A (5, 5)
D	B = C*D		COMPLEX A
	BB = A (I, J)**2		DOUBLE PRECISION B, C, D
	IF (B) 5, 5, 10		BB = PART (A, A (I, J), 25)**2
	5 CALL XYZ (A, B, C**2)		IF (SNGL (B)) 5, 5, 10
	10 AB = C*D - A - SINF (A)	5	CALL XYZ (A, B, SNGL (C)**2)
		10	AB = SNGL (C) *SNGL (D) - REAL (A) -SIN (A)

In construction of the PART function in the example above, if J $\leq$ 5, the arithmetic statement is equivalent to:

```
BB = REAL (A (I, J) )**2
```

But if J > 5, it is equivalent to:

```
BB = IMAG (A(I,J) )** 2
```

2. On left side of arithmetic statement--If a reference similar to that described above occurs on the left side of an arithmetic statement, then the statement specifies that a quantity be stored in one part of the number pair. To accomplish this operation in FORTRAN IV, GIFT replaces the arithmetic statement by a call to the STORE subroutine.

   The first three arguments are the same as in a call to PART, and the fourth argument is the expression whose value is to be stored. For example:

FORTRAN II	GIFT Translation
I DIMENSION A (5, 5, 5)	DIMENSION A (5, 5, 5)
A (I, J, K) = B**2	COMPLEX A
	CALL STORE (A, A (I, J, K), 125, B**2)

- Floating-Point Constants. In double-precision statements all floating-point constants are made double-precision by the use of a letter D followed by an exponent. Thus, for example:

FORTRAN II	GIFT Translation
D A = 1567.0 E 15 * X+2.4	DOUBLE PRECISION A, X
	A = 1567.0 D 15*X+2.4D0

- Constants in Single-Precision Statements. In single-precision statements, constants containing more than nine significant digits are truncated to nine significant digits so as not to be taken as double-precision in FORTRAN IV. For example:

FORTRAN II	GIFT Translation
X = 1234567895.	X = .123456789 E + 10

- Floating-Point Constants Ended With Non-Numerics. All floating-point constants whose last character is a non-numeric have a zero appended when they are translated. For example:

FORTRAN II	GIFT Translation
X = 2.4E	X = 2.4E0
D Y = 2.4E	Y = 2.4D0

COMMON STATEMENTS

- Manufactured COMMON Statements. Because EQUIVALENCE statements no longer affect COMMON storage, GIFT manufactures a COMMON statement, inserting dimensioned artificial variables where necessary, to preserve the order of COMMON storage. For example:

FORTRAN II	GIFT Translation
COMMON A, B, C, D EQUIVALENCE (B (2), E)	COMMON B, Q000CM (1), A, C, D EQUIVALENCE (B (2), E)

Note that the manufactured COMMON statement includes dimension information (but only for the artificially inserted variables), as allowed in FORTRAN IV.

- Addition of Undimensioned Artificial Variables. If in COMMON storage a double-precision or complex COMMON variable would begin in an odd-numbered memory location, an undimensioned artificial variable is inserted in the COMMON statement to cause the variable to begin in an even-numbered location. For example:

FORTRAN II	GIFT Translation
COMMON R, S, T	DOUBLE PRECISION S, X, Y
D S = X + Y	COMMON R, Q001CM, S, T

A similar insertion is made, if possible, when a double-precision or complex variable related to a COMMON variable through an EQUIVALENCE statement would begin in an odd-numbered location. However, certain cases of this type cannot be translated. See page 17 for further details.

- Modification of EQUIVALENCE Statements. An EQUIVALENCE Statement involving a double-precision or complex array must be modified, since subscripts in such equivalences are interpreted differently in FORTRAN II and FORTRAN IV. For example, in FORTRAN II the following EQUIVALENCE statement specifies that E shares storage with the imaginary part of A, assuming that A is not dimensioned:

```
      EQUIVALENCE (A (2), E)
D     A = B**2
```

In FORTRAN IV, an EQUIVALENCE reference to A (2) is interpreted as referring to the first word of the second number pair. To achieve the same storage allocation as the FORTRAN II program, GIFT (1) creates an insert variable, (2) generates an EQUIVALENCE between the insert variable and the double-precision or complex variable, and (3) replaces each occurrence of the latter variable in an EQUIVALENCE statement by the insert variable. Occurrences of the variable outside of EQUIVALENCE statements are not altered. If, for example, the insert variable Q000EQ is chosen to replace A, GIFT translates the last example as follows:

```
DOUBLE PRECISION A, B
EQUIVALENCE (A, Q000EQ)
EQUIVALENCE (Q000EQ (2), E)
A = B**2
```

- Treatment of Repeated Variables. In FORTRAN II, a variable can appear more than once in a COMMON statement. This situation is illegal in FORTRAN IV, and GIFT always reconstructs such COMMON statements. For example:

FORTRAN II	GIFT Translation
COMMON A, B, C, A, D, E	COMMON A, B, C, D, E

Arithmetic Statement Functions

In FORTRAN II, argument variables to arithmetic statement functions are dummies. No conflict arises if the same variable names appear as arguments to arithmetic statement functions and also as genuine variable names in the body of the program. In FORTRAN IV, a conflict may arise involving Type statements. Consider the following FORTRAN II example:

```
D     DIMENSION X (10, 10)
      FIRSTF (X, I) = A + X**I
```

The variable X in the DIMENSION statement is a double-precision variable. The variable X in the arithmetic statement function definition is a single-precision dummy. However, in FORTRAN IV any Type statement applied to X holds wherever X is used, even in a dummy argument. GIFT anticipates this possible conflict by replacing all dummy arguments in arithmetic statement functions with insert variables. Thus, if the insert variable Q000FL is chosen to replace X, and K000FX is chosen to replace I, GIFT translates the last example as follows:

```
DIMENSION X (10, 10)
DOUBLE PRECISION X
FIRST (Q000FL, K000FX) = A+Q000FL**K000FX
```

DIMENSION Statements

To ensure that every array is mentioned in a DIMENSION statement before it appears in an executable statement, every DIMENSION statement is relocated near the beginning of the program.

Hollerith Literals

Each Hollerith literal in an arithmetic or IF statement is replaced by an artificial variable name; and a DATA statement is manufactured, specifying the variable name and its corresponding value. For example (where b denotes a blank):

FORTRAN II	GIFT Translation
WORD = 4HbEND	DATA Q000HL/6HbENDbb/ WORD = Q000HL

Hollerith arguments in CALL statements are not altered.

Implicit Multiplication

Certain cases of implicit multiplication are allowed in FORTRAN II. GIFT inserts an * wherever necessary to remove the ambiguity in accordance with the rules for FORTRAN IV. For example:

FORTRAN II

```
X = (A + B) C + 3.D - (E + F) (G + H) 5. (U-V)
```

GIFT Translation

```
X = (A + B) * C +3. *D - (E + F) * (G + H) *5. * (U-V)
```

Variables with Too Few Subscripts

The FORTRAN II compiler accepts singly-subscripted references to multiply-dimensioned variables. When GIFT encounters such a reference in statements other than EQUIVALENCE statements, it appends sufficient trailing subscripts (with value 1) to make the number of subscripts in the reference equal to the number of dimensions in the DIMENSION statement. Trailing subscripts are also added to unsubscripted references to dimensioned variables in arithmetic, IF, and CALL statements, except when they are explicit arguments to functions or to CALL statements. For example:

FORTRAN II

```
DIMENSION X (10, 10, 5), Y (10, 10), Z (10, 10)
EQUIVALENCE (X (3), XX)
X (I) = Y (I) *Z-SINF (Y) - COSF (Y (3))*SINF (Y**2)
CALL XYZ (X, Y (3), Z**2)
```

GIFT Translation

```
DIMENSION X (10, 10, 5), Y (10, 10), Z (10, 10)
EQUIVALENCE (X (3), XX)
X (I, 1, 1) = Y (I, 1) *Z (1, 1) - SIN (Y) - COS (Y (3, 1) )
CALL XYZ (X, Y (3, 1), Z (1, 1)**2)
```

Unsubscripted references to dimensioned variables are unchanged.

DO Statement Indexing Parameter

Several incompatibilities exist between FORTRAN II and FORTRAN IV in regard to the indexing parameter of a DO statement or an implied DO in input/output lists. These inconsistences, which cannot be resolved by GIFT and must be corrected by hand, are listed below:

1. In FORTRAN II, the name of the indexing parameter within a DO loop may be the same as the name of a dimensioned fixed-point variable used outside the DO loop.

   FORTRAN IV, however, objects to this and issues a level 3 error message which suppresses assembly. The following is an example of this incompatibility:

```
      DIMENSION J (10, 2)
      .
      .
      .
      .
      DO 10 J = 5, 100, 5
      LL = J
10    PRINT 35, LL
      .
      .
      .
      .
      END
```

2. In FORTRAN II, if the program contains a storage location having the same fixed-point variable name as the indexing parameter of a DO (or an implied DO) statement, the storage location is not affected after a normal exit unless the indexing parameter was used as a variable within the DO range or it was used as a subscript in combination with a relative constant whose value changes within the range of the DO statement. In FORTRAN IV, however, the storage location is always updated and (after a normal exit from the DO range) contains the highest value of the indexing parameter.

MODIFICATIONS FOR FORMAT CONFORMITY

READ/WRITE Statements

READ/WRITE statements are modified according to the FORTRAN IV specification. For example:

FORTRAN II	GIFT Translation
READ INPUT TAPE IN, 100, A, B, C	READ (IN, 100) A, B, C
WRITE TAPE IOUT, D, E, F	WRITE (IOUT) D, E, F

At the option of the user, references to file numbers can be replaced either by other variable names or by file constants by means of an *REPLACE card. (p. 22) If the replacements are variables, a DATA statement is constructed assigning the proper values to the variables, and INTEGER Type statements are manufactured for these variable names if they begin with A through H or with O through Z. For example:

FORTRAN II	GIFT Translation
*REPLACE (6, AOUT), (3, 5) WRITE OUTPUT TAPE 6, 120, G, H, I REWIND 3	INTEGER AOUT DATA AOUT/6/ WRITE (AOUT, 120), G, H, I REWIND 5

READ DRUM/WRITE DRUM statements are flagged as errors and are not translated by GIFT.

Double-precision and complex variables included in the list of input/output statements are flagged by GIFT, but the statements are not considered in error and are translated.

Statements Concerning Internal Switches

Each statement which sets or tests an internal switch (sense switch, overflow indicator, etc.) is translated into a call to an appropriate subroutine. This is followed, in the case of tests, by a computed

GO TO statement. For example, assume an insert variable with the name K000FX. Then the following set of FORTRAN II statements is translated as shown:

FORTRAN II

```
100  SENSE LIGHT 3
110  IF (SENSE LIGHT 2) 111, 112
120  IF (SENSE SWITCH 5) 121, 122
130  IF DIVIDE CHECK 131, 132
140  IF ACCUMULATOR OVERFLOW 141, 142
150  IF QUOTIENT OVERFLOW 151, 152
```

GIFT Translation

```
100  CALL SLITE (3)
110  CALL SLITET (2, K000FX)
     GO TO (111, 112), K000FX
120  CALL SSWTCH (5, K000FX)
     GO TO (121, 122), K000FX
130  CALL DVCHK (K000FX)
     GO TO (131, 132), K000FX
140  CALL OVERFL (K000FX
     GO TO (141, 142), K000FX
150  CALL OVERFL (K000FX)
     GO TO (151, 152), K000FX
```

FORMAT Generator

GIFT recognizes sets of FORMAT Generator cards and leaves them unaltered.

RIT and WOT

GIFT recognizes RIT and WOT as equivalent to READ INPUT TAPE and WRITE OUTPUT TAPE, respectively, and translates them as explained on p. 12.

Characters in Column 1

Except on FORMAT Generator cards, nonnumeric characters other than *, $, and C are removed from column 1. Every * or $ is replaced by a C, and all comment cards remain unchanged.

FAP Programs

Programs preceded by a *FAP card are ignored by the translator. The end of the FAP program is recognized when the number of FAP END cards encountered exceeds the number of MACRO and MOP cards encountered. (ENDM is recognized as an END card for this purpose.) Inclusion of FAP UPDATE decks in GIFT input causes errors if the UPDATE decks do not contain the proper number of END cards.

$ DATA Card

GIFT recognizes a $ DATA card and considers all subsequent cards as data until it encounters either a FORTRAN or a GECOS control card ($ in column 1). Data cards are listed as part of the input programs but are ignored by the translator.

2. PROGRAM REQUIREMENTS AND USAGE

GIFT is a program written in FORTRAN IV and macro assembly language that is designed to run under control of standard GE-635 software. The subprograms to be translated are considered data and are placed behind a $ GIFT control card in the deck. When the job is run, output is written on the SYSOUT file.

REQUIREMENTS

Characteristics of Subprograms to be Translated

A subprogram that is to be translated must fulfill the following requirements:

1. It must be programmed in the FORTRAN II language and be capable of being compiled successfully by the IBM 7000 Series FORTRAN II Compiler. GIFT does little diagnostic checking, and an incorrect translation may result for a program which can not be compiled by the FORTRAN II system.

2. It must terminate in the normal manner with the FORTRAN END card.

3. If the first card in the input source deck has a C in column 1, then columns 2-5 are punched in columns 73-76 of the output source program with zeros completing the field when necessary. If this comment card is missing or card columns 2-5 are blank, the subroutine name (or function name) will be used for a subprogram and a zero field used for the main program. The output source deck is sequenced by tens in columns 77-80, recycling to zero when the field reaches maximum value.

   All labeling and sequencing in the input source program is replaced in the above manner.

Any $ control cards included before the first statement of the FORTRAN II deck, however, are labeled with zeros in columns 73-79 and sequenced by 2 in column 80, starting with 1-9 and continuing from A through Z.

Table Sizes

The following table sizes are allowed:

1. GIFT allows for a total of 4000 COMMON, DIMENSION, EQUIVALENCE, double-precision, and complex variables in a single program or subprogram. Currently FORTRAN II allows 2400 COMMON, 1160 DIMENSION, 6000 EQUIVALENCE (that is, literal appearances), and 550 double-precision and complex variables--a total of 10110. Thus, GIFT can handle 6110 fewer variables than the maximum case.

2. GIFT allows for replacement of a maximum of 20 different variable names in any one program under the REPLACE option.

3. GIFT provides for a maximum of 100 library, built-in, and arithmetic statement function names beginning with X or letters from I through N.

4. GIFT allows a maximum of 100 function names appearing on F cards.

Chain Jobs

No attempt is made to translate chain jobs automatically. GIFT flags the appearance of a CALL CHAIN statement by inserting a comment card containing asterisks in columns 2-72.

Double-Precision and Complex EQUIVALENCE Variables

- Odd-Even Memory Storage Inconsistencies. Because the most significant part of a double-precision variable--or the real part of a complex variable--must be stored in an even-numbered memory location, certain EQUIVALENCE specifications may result in odd-even inconsistencies. For example, consider the following:

```
      EQUIVALENCE (A, B), (A (6), C)

D     G = B + C
```

  Clearly, the more significant parts of B and C cannot both begin in an even-numbered memory location. In such cases, GIFT prints out a diagnostic message. The inconsistency must be resolved manually and the revised program retranslated.

- Implied Synonym Relationships. FORTRAN II stores the imaginary parts of a complex array--or the low-order parts of a double-precision array--in a separate block, while FORTRAN IV stores the two parts of each array element in consecutive memory words. GIFT maintains the relative positions of arrays linked through EQUIVALENCE statements, but it does not always maintain implied synonym relationships. For example:

```
I     DIMENSION A (5)

      EQUIVALENCE (A (2), B)
```

  This FORTRAN II expression specifies that B shares storage with the second word of the 10-word block A. These statements also imply that B shares storage with the real part of the second complex number in A. When the FORTRAN IV compiler processes the GIFT/Translated version of this program, B still shares storage with the second word of block A; but B also implicitly shares storage with the imaginary part of the first complex number in A.

Double-Precision and Complex COMMON Variables

An equivalence relationship implied through different COMMON statements in two or more programs of a job may be destroyed if one of the COMMON statements contains double-precision or complex variables. For example:

```
      SUBROUTINE SUB1          SUBROUTINE SUB2
      COMMON R, S, T           COMMON X, Y, Z
I     A = S**2                 DIMENSION Y(2)
```

In this example, R is equivalent to X, S to Y, and T to Z. The beginning of the GIFT/Translated version of SUB1 would be:

```
SUBROUTINE SUB1
COMMON R, Q000CM, S, T
COMPLEX S, A
A = S**2
```

The artificial variable Q000CM is inserted in COMMON to cause the real part of the complex variable S to be stored in an even-numbered location. The implicit equivalence relationship between S and Y and between T and Z is thus destroyed.

GIFT processes each subprogram of a job separately and therefore does not modify the COMMON statement of SUB2 in the above example.

Continuation Cards

The GIFT/Translated version of certain statements is sometimes longer than the original statement. In such cases, GIFT generates up to 19 continuation cards. If a statement requires more than 20 such cards, a diagnostic message is written and the statement is ignored.

USAGE

Deck Setup

Any number of source programs can be translated during one run of GIFT. Each such source program must be preceded by a $ GIFT card and be followed by an END card.

FAP programs need not be deleted from FORTRAN II decks, since the *FAP card causes such programs to be ignored by GIFT.

An end-of-file mark on the input file (automatically supplied) signifies the end of the job and terminates GIFT execution.

If a programmer changes the input file by using a file control card (such as a GECOS $ TAPE or $ DISC card), it must be placed behind the $ GIFT card in the GIFT program; and the remainder of the program must be placed on the input file specified.

Programmer Option Cards

For most translation jobs, no special options need be specified. However, file control, *ASSIGN, and *REPLACE cards allow the user to modify the GIFT file assignments, the insert variables, and the existing variables in the source program.

- File Control Cards--Altering File Assignments. In addition to required system files, GIFT requires the files listed below during the translation of a subprogram:

  F_1--Input file
  F_2--Output file
  F_3, F_4, F_5--Intermediate or scratch files

  Usually, any modification of file assignments is permanent and is performed by an installation for its particular requirements. The user can modify the assignment of specific physical devices for files by means of file control cards as described in Chapter III of the GE-635 General Comprehensive Operating Supervisor Reference Manual, CPB-1002. The *ASSIGN card described below is used to define the logical files to be used by GIFT.

- *ASSIGN Cards--Changing Insert Variable Names. During the process of subprogram translation, it is sometimes necessary to replace the names of variables and constants in the source program with new ones and to insert dummy variables in

COMMON statement translations. For instance, the following statement could appear in the source program:

```
READ INPUT TAPE 5, 20, A, B
```

This might then be translated into:

```
INTEGER Q000TP
DATA Q000TP/5/
READ (Q000TP, 20) A, B
```

To make the insert variable names as unique as possible, the following list is used by GIFT.

Insert Variable Name	Use
Q000TP	To replace numeric tape references
Q000CM	To use as dummy inserts in COMMON
Q000HL	To replace Hollerith literals in arithmetic statements
Q000CT	To replace octal constants in Boolean statements
Q000FL	To replace or insert a floating-point variable
K000FX	To replace or insert a fixed-point variable
Q000EQ	To replace double-precision or complex equivalences
QQ000Q, Z000QQ, Z00Q0Q Z0Q00Q, ZQ000Q, K000ZQ Q00Z0Q, Q0Z00Q, QZ000Q Q000QZ, K00Q0Z, Q0Q00Z QQ000Z	Spares for all above

Numeric file references are not replaced by GIFT unless a file variable replacement name was given as the first variable name on the *ASSIGN card currently in effect. Individual file constants may still be replaced by use of *REPLACE card.

The user who wishes to replace one or more of these variable names by using an *ASSIGN card must be familiar with the method used by GIFT when selecting an insert variable. This is best described by an example.

Suppose that four octal constants in a Boolean statement must be replaced by variable names. The names selected by GIFT might be (in succession from left to right):

Q000CT Q001CT Q002CT Q003CT

Construction of insert variables would continue in this manner until Q999CT was reached. If more replacements were necessary, one of the spares would be taken and the modification procedure would begin anew.

Replacement names are assigned in the order of appearance of the item to be replaced in the input source program. Therefore, file constants, Hollerith literals, and octal constants may be assigned different replacement names in different subprograms.

The following rules apply when changing insert variables in the list:

1. If the new variable contains fewer than six characters, it is left-adjusted and filled with trailing zeros.

2. The new variable should contain some numeric characters so that incrementing of the characters can occur.

3. If the Hollerith literal, octal constant, or floating-point replacement is changed, it must not begin with any letter from I through N.

4. If the fixed-point replacement is changed, it must begin with any letter from I through N.

The *ASSIGN card--used for changing logical file assignments and/or inserting variable names, as just discussed--should be prepared and used in the following manner:

1. Format:

Column 1 7

* ASSIGN (F_1, F_2, ... F_5) V_1, V_2, ... V_n

Where:

F_1, F_2, ... F_5 refer to the respective files listed in the table on page 19.

V_1, V_2, ... V_n refer to the respective insert variable names listed in the table on page 20.

Blanks are ignored.

Example: To change the output file to file 11 and to change the file constant and floating-point variable replacements:

```
*    ASSIGN (, 11) K00T....AAX009
```

2. Placement: The *ASSIGN card must be the first card in the source deck to be translated.

3. Persistence: An *ASSIGN card remains in effect until changed by another *ASSIGN card. To return to the original GIFT configuration, a blank *ASSIGN card may be used.

- *REPLACE Card. The user may change the variable names used in a source program by means of the *REPLACE card. Also, it may be used to replace a file designator constant with a variable or another constant.

  For example, it may be desirable to change the name of a table of fixed-point items from ITABLE to TABLE. If the replacement is requested via the *REPLACE card, the translator generates the statement INTEGER TABLE and changes all references of ITABLE to TABLE. A maximum of 20 variable names may be replaced.

  The card should be prepared and used in the following manner:

  1. Format:

     Column 1 7

     * REPLACE $(A_1, B_1), (A_2, B_2), \ldots (A_n, B_n)$

Where:

> $A_1, A_2, \ldots A_n$ are the variable names or file constants to be replaced and $B_1, B_2, \ldots B_n$ are the replacement names or constants.

Blanks are ignored.

Example: A *REPLACE card is punched as follows:

*REPLACE (S, T), (T, S)

> This changes the statement S = T in the source program to T = S in the translation.

2. Placement: *REPLACE cards must follow directly behind the *ASSIGN card or be the first cards in the source program if there is no *ASSIGN card.

3. Persistence: A *REPLACE card affects only a single input program.