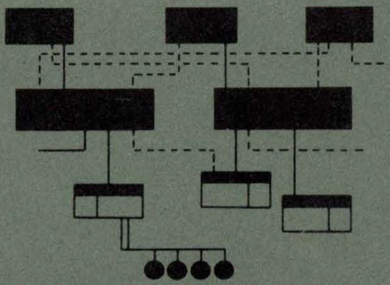
GE-625/635 GECOS-III Time-Sharing FORTRAN

Information Systems

Information Systems Equipment



GENERAL ELECTRIC

CPB-1566

CPB-1566

GE-625/635 GECOS-III Time-Sharing FORTRAN

December 1968

INFORMATION SYSTEMS

GENERAL ELECTRIC

Preface

The intent of this manual is to supply reference material pertaining to the use of Time-Sharing FORTRAN, a system of the GE-625/635 GECOS-III Time-Sharing System.

The manual is divided into two sections -- time-sharing system operation and FORTRAN language characteristics. The former is concerned with communications with the time-sharing system and the storing and modification of files. The latter describes the methods for writing computer programs in the FORTRAN language for the time-sharing system.

This manual includes new features implemented in Systems Development Letter 1.

A portion of the material presented in this manual is reprinted by permission from IBM 7090/7094 Programming Systems: FORTRAN IV Language (Reference Manual C28-6274-1), © 1963 by International Business Machines Corporation.

This manual was produced using the General Electric Remote Access Editing System (RAES). RAES is a time-shared disc-resident storage and retrieval system with text-editing and manuscript formatting capabilities. The contents of the manual were entered into RAES from a remote terminal keyboard, edited using the system editing language, and formatted by RAES on reproduction masters.

The index was produced using a computer-assisted remote access indexing system. This system produces an index using source strings delimited at manuscript input time.

Suggestions and criticisms relative to form, content, purpose, or use of this manual are invited. Comments may be sent on the Document Review Sheet in the back of this manual or may be addressed directly to Documentation, B-107, Processor Equipment Department, General Electric Company, 13430 North Black Canyon Highway, Phoenix, Arizona 85029.

(2.5M 12-68)

Contents

1. Introduction

Time-Sharing FORTRAN, a system of the GE-625/635 Comprehensive Operating Supervisor (GECOS-III) Time-Sharing System, is an engineering and scientific language designed for the solution of mathematical problems. Time-Sharing FORTRAN is a version of FORTRAN IV; differences between Time-Sharing FORTRAN and FORTRAN IV are listed in Appendix C.

The time-sharing system enables a large number of individuals to make use of ("time-share") a GE-625/635 Information System simultaneously. The user works directly with the computer, whether it is within his sight or miles away, with the belief that he has exclusive use of the computer, even though many others are making simultaneous use of the computer. The Time-Sharing FORTRAN user submits his program, controls its compilation and execution, and upon request, supplies data input, all from a terminal device.

Time-Sharing FORTRAN encompasses the file input/output (e.g., disc and drum file) capabilities of a "batch-world" FORTRAN processor, besides permitting direct, input/output by the way of the user's terminal. The terminal device can be one of a variety of units, but the assumption made for the purpose of this manual is that the terminal is a teletypewriter, with or without a paper tape reader/punch. The information concerning the teletypewriter is generally applicable to other types of terminals.

Time-Sharing FORTRAN makes use of the GECOS-III File System, a system common to the three dimensions of GE-625/635 GECOS-III processing: batch, remote batch, and time-sharing. A significant advantage of this commonality is that the FORTRAN user can refer to and process collections of data--in proper format--created and/or maintained in the batch or remote batch environments. From the user's standpoint, the file system is a device for readily storing sets of information which may either be programs that specify instructions to be executed by the computer or data which is to be read or written by a user's programs. Such sets of information are called files. These files, upon request of the user, are stored permanently on a disc or drum file and are always referred to by name. The name, size, and type of each user's files are maintained in a user's master catalog identified by the user.

SECTION I

TIME-SHARING SYSTEM

2. Time-Sharing-System Operation

GENERAL OPERATION

The standard means of communication with the GE-625/635 GECOS-III Time-Sharing System (TSS) is by way of a teletypewriter used as a remote terminal. Other compatible devices may also be used, but use of a teletypewriter is assumed in this manual. The user may choose either the keyboard/printer or paper-tape teletypewriter unit for input/output, or combine both. In either case, the information transmitted to and from the system is displayed on the terminal-printer. Keyboard input will be used for purposes of description; instructions for the use of paper tape are given under "Paper Tape Input" in this chapter.

The user "controls" the time-sharing system primarily by means of a command language, a language distinct from any of the specialized programming languages that are recognized by the individual time-sharing compilers/processors (e.g., the Time-Sharing FORTRAN language, in this case). The command language is, for the most part, the same for users of any component of the time-sharing system; i.e., FORTRAN, BASIC, Text Editor, etc. A few of the commands pertain to only one or another of the component time-sharing systems, but the majority of them are, in form and meaning, common to all component systems.

The commands relate to the generation, modification, and disposition of program and data files, and program compilation/execution requests. Commands that pertain particularly to the use of Time-Sharing FORTRAN are described in this chapter. The complete time-sharing command language is described in Appendix A, in reference form.

Once communication with the system has been established, any question or request from the system must be answered within ten minutes, except for the initial requests for user identification (user-ID) and sign-on password, which must be given within one minute. If these time limits are exceeded, the user's terminal will be disconnected.

LOG-ON PROCEDURE

To initiate communication with the time-sharing system, the user performs the following steps:

- Turns on the terminal unit
- Obtains a dial-tone
- Dials one of the numbers of his time-sharing center

The user will then receive either a high-pitched tone indicating that his terminal has been connected to the computer or a busy signal. The busy signal would indicate, of course, that no free line is presently available.

Once the user's terminal has been connected to the computer, the time-sharing system begins the log-on procedure by transmitting the following message:

```
THIS IS THE GE-600 TSS SYSTEM ON(date)AT(time)CHANNEL(nnnn)
```

where time is given in hours and thousandths of hours (hh.hhh), and nnnn is the user's line number.

Following this message, the system asks for the user's identification:

```
    USER ID --
```

The user responds, on the same line, with the user-ID that has been assigned to him by the time-sharing installation management. This user-ID uniquely identifies a particular user already known to the system, for the purposes of locating his programs and files and accounting for his usage of the time-sharing resources allocated to him. An example request and response might be:

USER ID -- <u>J.P.JONES</u>

Note

A carriage return must be given following any complete response, command, or line of information typed by the user.

(The user's response is underlined here for illustration.) After the user responds with his user-ID, the system asks for the sign-on password that was assigned to him along with his user-ID, as follows:

```
    PASSWORD
    XXXXXXXXXXXX
```

The user types his password directly on the "strikeover" mask provided below the request PASSWORD. The password is used by the system as a check on the legitimacy of the named user. The "strikeover" mask insures that the password, when typed, cannot be read by another person. (In the event that either the user-ID or password is twice given incorrectly, the user's terminal is immediately disconnected from the system.) At this point, if the accumulated charges for the user's past time-sharing usage equals or slightly exceeds 100% of his current resource allocation, he will receive a warning message. If his accumulated charges exceeds 110% of his current resources, he receives the message:

```
RESOURCES EXHAUSTED - CANNOT ACCEPT YOU
```

and his terminal is immediately disconnected. (The user may also receive the following information message if his situation warrants it:

```
*ALLOTTED FILE SPACE > 88% USED
```

This condition does not affect the log-on procedure.)

Assuming that the user has responded with a legitimate user-ID and password and has not overextended his resources, the time-sharing system then asks the user to select the processing system that he wants to work with; this is called the system-selection request. In this case, the user would respond with FORTRAN:

```
SYSTEM ? FORTRAN
```

The user is then asked whether he now wants to enter a new program (NEW) or if he wants to retrieve and work with a previously entered and saved program (OLD); the request message is:

```
OLD OR NEW -
```

If the user wishes to start a new program (i.e., build a new source file), he responds simply with:

```
NEW
```

If, on the other hand, he wants to recall an old source-program file, he responds with:

```
OLD filename
```

where filename is the name of the file on which the old program was saved during a previous session at the terminal (see the SAVE command).

Following either response, the system types the message:

```
     READY FOR INPUT
```

returns the carriage, and prints an asterisk in the first character position of the next line:

```
     READY FOR INPUT

     *
```

An example of a complete log-on procedure, up to the point where the FORTRAN system is ready to accept program input or control commands, might be as follows:

```
THIS IS THE GE-600 TSS SYSTEM ON 07/26/68 AT 14.768 CHANNEL 0012

USER ID  - J.P.JONES
PASSWORD
XXXXXXXXXXXX      - (user's password is typed over the mask)
SYSTEM ? FORTRAN
OLD OR NEW - NEW  - (NEW is shown arbitrarily for illustration)
READY FOR INPUT
*                 - (the user begins entering input on this line)
```

ENTERING PROGRAM-STATEMENT INPUT

After the message:

```
     READY FOR INPUT
```

the system is in build-mode (as indicated by the initial asterisk) and is ready to accept FORTRAN program-statement input or control commands (see below for description of the commands). All lines of input other than control commands are accumulated on the user's current file. Normally the current file will be the file that contains the program he wants to compile and run at this session. If he is building a new file (NEW response to OLD OR NEW--), his current file will initially be empty. If he has recalled an old file (OLD filename) the content of the named old file will initially be on his current file, and any input typed by the user -- excepting control commands -- will be either added to, merged into, or will replace lines in the current file, depending upon the relative line numbering of the lines in the file and the new input. (This process is explained under the heading "Correcting or Modifying a Program," below.)

Following each line of noncommand-language input and the terminating carriage response, the system will supply another initial asterisk, indicating that it is ready to accept more input.

Format of Program-Statement Input

A line of FORTRAN input -- as distinct from a control command -- can contain one of the following:

a. One or more FORTRAN statements.

b. A partial statement containing at least a statement identifier.

c. A continuation of a statement left incomplete in the preceding line of input.

d. A comment (to be included in the source-language program listing).

e. A combination of (c) and (a) or (b), in that order.

A line of input must begin with a line-sequence number of from one to eight numeric characters; this number may optionally be preceded by one or more initial blanks. The line-sequence number facilitates correction and modification of the source program (described below); hereinafter, the line-sequence number will be referred to simply as the "line number." (Note that a line number is distinct from a statement number; a statement number is a part of the FORTRAN-language statement itself.)

The line number is always terminated (i.e., immediately followed by) with a single control character which may be a blank, an ampersand, an asterisk, or the letter C. The control character merely serves to indicate what type information is to follow (new statement, continuation, or comment) and is not compiled as part of the program.

The semicolon may be used to indicate the end of one complete FORTRAN statement and the beginning of another on the same line of input. A carriage-return must, of course, be used to terminate a complete line of input.

The general format of a line of FORTRAN input is then as follows (square brackets indicate optional portions of the format):

nnnnnnnnc statement or continuation [;statement...;statement]

or

nnnnnnnnc comment

where:

<u>nnn...n</u> is a one-to-eight character, numeric line number and

<u>c</u> is a single-character control character which may be a blank, an ampersand, an asterisk, or the letter C, and must immediately follow the last digit of the line number.

<u>Significance of the Control Character</u>. The control character identifies the type of information that follows it.

b̸ (blank) -- if the character position immediately following the last digit of the line number contains a blank, the next nonblank character is assumed to begin a new FORTRAN statement. In this case, the next nonblank character may begin a FORTRAN statement number (i.e., mm...m statement-text).

& (ampersand) -- if an ampersand terminates the line number, the next nonblank character is assumed to be a continuation of the (last) statement in the previous line of input. The effect of "&" is to suppress the previous carriage return as an end-of-statement indicator.

* (asterisk)

<u>or</u>

C -- if the line number is terminated with an asterisk or the letter C, the information following is assumed to be a comment. The comment itself is terminated by a carriage return.

A semicolon within a noncomment line indicates both the end of the preceding statement and that any significant information (nonblank, noncarriage return) following it begins a new statement. The new statement may include a FORTRAN statement number, mm...m.

The format of a statement, as typed in following a blank control character, is:

...nnb̸ <u>b̸...b̸ mm...m FORTRAN-language text</u>

(The statement-format portion is underlined.)

where:

<u>b̸...b̸</u> are optional blanks, and

<u>mm...m</u> is an optional numeric statement number (no practical limit on its size).

<u>Blanks (or Spacing) Within a Line of Input.</u> Initial, imbedded, or trailing blanks in a line of input have no significance in its interpretation, excepting only that blanks are illegal within the line number and that the nonnumeric character (including b̸) immediately following the line number is interpreted as a control character. Thus, spacing can be used quite freely within a line of input in the interests of legibility. (Blanks with ASCII/filename constants and nH fields -- i.e., alphanumeric information -- are meaningful; however, they are retained in the object program coding.)

Note that the Time-Sharing FORTRAN language is, therefore -- except for the relative position of the control character -- completely free-form, or position independent.

Correcting or Modifying a Program

Keyboard input is sent to the computer and written onto the user's current file in units of complete lines. A line of terminal input is terminated by a carriage return and no part of the line is transmitted to the system until that carriage return is given. Therefore, corrections or modifications can be done at the terminal at two distinct levels:

- Correction of a line-in-progress (i.e., a partial line not yet terminated).
- Correction or modification of the program (i.e., the contents of the user's current source file) by the replacement or deletion of lines contained therein, or the insertion of new lines.

The correction of a typing error that is perceived by the user before the line is terminated can be done in one of two ways. He may delete one or more characters from the end of the partial line or he may cancel the incomplete line and start over. The rules are as follows:

a. Use of the commercial "at" character (@) deletes from the line the character preceding the @ character; use of n consecutive @ characters deletes the n preceding characters (including blanks.)

   Examples:

   *ABCD$\underline{\text{F}}$@E would result in ABCDE being transmitted to the program file.

   *ABC$\underline{\cancel{\text{b}}\text{DEF}}$@@@@DEF would result in ABCDEF being transmitted. (The characters to be deleted are underlined for illustration.)

b. Use of the CTRL (control) and X keys, depressed simultaneously, causes all of the line to be deleted. The characters DEL are printed to indicate deletion and the carriage is automatically returned. For example:

```
*ACDEFG [CTRL/X] DEL          (all characters deleted)
*                             (ready for new input)
```

Correction or modification of the current source file is done on the basis of line numbers and proceeds according to the following rules:

a. Replacement. A numbered line will replace any identically numbered line that was previously typed or contained on the current file (i.e., the last entered line numbered nnn will be the only line numbered nnn in the file).

b. Deletion. A "line" consisting of only a line number (i.e., nnn) will cause the deletion of any identically numbered line that was previously typed or contained on the current file.

c. Insertion. A line with a line-number value that falls between the line-number values of two pre-existing lines will be inserted in the file between those two lines.

At any point in the process of entering program-statement input, the LIST command may be given, which results in a "clean," up-to-date copy of the current file being printed. In this way, the results of any previous corrections or modifications can be verified visually. (The several forms of the LIST command are described in detail below.) Following the response (or command) OLD filename, the LIST command can be used initially to inspect the contents of the current source file (i.e., the "old" program).

A comprehensive example of program correction and modification follows. Replies to the user from the system are underlined here;in actual use, no underlining is done. Explanations are enclosed in parentheses; they are not part of the printout.

```
USER ID -ED.W
PASSWORD--
XHEKKEKEHXXKK
SYSTEM  ?  FORTRAN
ØLD ØR NEW-NEW
READY FØR INPUT
*AUTØX - (enter automatic-line-number mode)
*0010     READ:A,B,C
*0020     X1=A*B/C
*0030     X2=A**2;B**2
*0040     ANS=X2/X1
*0050     PRINT 10,X1,X2, ASN@@@ANS - (typing error correction)
*0060 10  FØRMAT(1X,"X1=",F6.S@2,"X2=",F8.2,"ANS=",
*0070&    F6.2)
*0080     STØP
*0090     END
*0100 -   (end automatic mode by carriage return)
*0030     X2=A**2+B**2-C - (replacement of line 30)
*SAVE FØRT01
DATA SAVED--FØRT01

*LIST     - (display corrected program)
0010     READ:A,B,C
0020     X1=A*B/C
0030     X2=A**2+B**2-C
0040     ANS=X2/X1
0050     PRINT10,X1,X2, ANS
0060 10  FØRMAT(1X,"X1=",F6.2,"X2=",F8.2,"ANS=",
0070&    F6.2)
0080     STØP
0090     END

READY

*RUN - (run program)

= 3.2,10.5,2.2 - (type input data)
X1= 15.27X2= 118.29ANS= 7.75 - (output - correct,
                                but poor format)
PRØGRAM STØP AT 80
*0060 10 FØRMAT(1X,"X1=",F6.2," X2=",F8.2," ANS=", - (correct
                                              format statement)
*RUN

= 3.2,10.5,2.2
X1= 15.27  X2=  118.29  ANS=  7.75 - (improved output format)
PRØGRAM STØP AT 80
*SAVE FØRT01
DATA SAVED--FØRT01  - (corrected version of program saved)

*BYE - (finished)
**RESØURCES USED $  2.08, USED TØ DATE $  263.85= 27%
**TIME SHARING ØFF AT  15.421 ØN  10/10/68
```

Since the program compiled correctly and returned valid answers, the user terminates his connection with the computer by typing BYE.

COMMANDS

The time-sharing system commands that are particularly pertinent for use in control of the Time-Sharing FORTRAN system are:

- RUN -- causes a source or object program to be compiled and/or executed. A concatenation of several program files or file segments may be specified as program input. Saving of the resultant object program may be requested.
- LIST -- causes the contents of either the current file or specified permanent file to be listed at the user's terminal. Portions of a file can be selected, by means of line numbers. Several files or file segments can be concatenated for listing.
- RESEQUENCE -- causes the existing line numbers in the current file to be replaced by an ordered arithmetic sequence of line numbers. Either the standard sequence (010,020,030,...) or a specified sequence can be obtained.
- DELETE -- causes a specified portion of the current file to be deleted; portion is specified by means of line numbers.
- OLD -- causes the contents of a specified permanent (saved) file to be copied to the current file. The previous contents of the current file (if any exist) are lost. Several permanent files or file segments may be concatenated on the current file.
- NEW -- causes the current file to be reinitialized (i.e., cleared). Previous contents of the current file are lost.
- AUTOMATICX (or AUTOX) -- causes line numbers to be generated by the system automatically at the time the initial (build-mode) asterisk is printed. Either the standard line number sequence or a specified sequence can be obtained.

- SAVE -- causes the contents of the current file to be saved on a specified permanent file. Saving can be done on a pre-existent (i.e., predefined) file or a new permanent file can be implicitly created.
- GET -- causes specified permanent files to be accessed (i.e., names placed in the user's available file table (AFT)) for later reference by the object program (data files) or by other commands (generally program files). The GET function is analogous to the "opening" of files in certain other contexts. (The current file is not affected.)
- PERM -- causes a specified temporary file (generally a data file created by the object program) to be saved on a specified permanent file.
- REMOVE -- causes specified file names to be deleted from the user's AFT. The AFT is of finite length and may become full unless unneeded file names are removed.
- TAPE -- causes file-building input to be read from paper tape.
- DONE -- causes an exit from the FORTRAN system; control returns to the system-selection level (SYSTEM ?).
- BYE -- causes the user to be logged off of the time-sharing system; usage charges are computed and terminal is disconnected.

Most of the commands listed above have several forms. The primary, or most-used, forms of these commands are described in detail in this section. The more sophisticated forms are described in Appendix A. These forms are used for describing catalog/file structures in the file system, as opposed to file references simply by file name.

There are other, more general purpose, time-sharing commands that may be used with the FORTRAN system; for example: CATALOG, which requests a listing of a user's catalogs and/or files in the file system. These general purpose commands are also described in Appendix A.

Definition of Current File

The term "current file," as used in this manual, denotes the temporary file implicitly referred to, and affected by, the OLD and NEW commands. It is also the file implicitly referred to by the simplest (or only) forms of several other commands (e.g., LIST, RESEQUENCE, DELETE).

The current file is automatically assigned to each user and is the file on which all program-file building or modification takes place. All build-mode, statement input from the terminal affects the current file.

In certain cases (as described below), the current file can be explicitly referred to, in a list of file names, by the name "*".

Command Descriptions

The commands are described below in alphabetical order. The command "name" is given in its complete form. However, where the command "name" is longer than four characters (e.g., RESEQUENCE), only the first four characters need be given (e.g., RESE). (Note that AUTOX is a special case, as is explained below.)

• AUTOMATICX

Function

To initiate the automatic generation of line numbers by the system (automatic mode), so that the user need not type them himself while entering a new program via the terminal keyboard. The line numbers appear on the terminal printer copy immediately following each build-mode request for input (initial *), and, unless deleted, form part of the respective line on the program file, just as though the user had typed them.

Forms of the Command

(1) AUTOMATICX (or) AUTOX

Causes the automatic creation of line numbers at the point at which the automatic mode is entered (or re-entered), with line numbers initially starting at 0010 and incrementing by 10. If automatic mode is cancelled and then re-entered, the next line number issued will be in sequence with the previous automatically-generated numbers.

(2) AUTOMATICX n,m (or) AUTOX n,m

Causes the automatic creation of line numbers, as in form (1), but starting with line number n and incrementing by m. The user specifies in this form a starting value (n) and the increment (m). The parameters n and m must be positive decimal integers.

Note

Using the forms given above, the line numbers will be issued without a terminating blank in the control-character position.

Usage

This command could be used before beginning to build a new file (i.e., following the command NEW). It might also be used following an OLD response or command. The automatic mode is canceled by a carriage return given by the user immediately following a line number. (That line number is also deleted from the file.) If the user wishes to replace a preceding line or insert a line while in automatic mode, he gives an immediate carriage return following the next-issued line number and performs his desired corrections in manual mode. He may then re-enter automatic mode at the point at which he canceled simply by giving AUTOX again. For example:

```
READY FOR INPUT
*AUTOX
*0010ƀREAL...
*0020ƀPRINT...
*0030ƀPRINT...
     .
     .
     .
*0170ƀDO...
*0180                  -(auto. mode canceled)
*0030ƀTOTAL            {corrections in manual mode}
*0035ƀREAD
*AUTOX                 -(auto. mode re-entered)
*0180ƀIF...
     .
     .
     .
*0340ƀEND
*0350                  -(auto. mode canceled, end of input)
```

Additional example:

```
*AUTOX 300,5
*0300ƀ100 CONT...
*0305ƀ200 MAT...
*0310ƀSTOP
    .
    .
    .
*0425                    -(auto. mode cancelled)
    .
    .
    .
*AUTOX
*0425                    -(auto. mode re-entered)
```

When a line-number value of 9999 is exceeded, the 4-place format is replaced by an 8-place format (i.e., 00010000, etc.).

The line-number counters (current value and increment) are reset only by a AUTOX n,m command.

Note that the "normal" form of this command is AUTOMATIC (or AUTO), which causes a terminating blank to be issued following the line number (i.e., nnnƀ). But since this blank occupies the FORTRAN control-character position (which may be blank, * or C, or &), the form AUTOMATICX or AUTOX, which suppresses the automatically-generated blank, is preferred for FORTRAN usage. If, however, the user does not intend to enter anything but complete statements (no comments or continuation lines), he may of course use the AUTO form, which will supply the blank control character automatically.

- BYE

Function

To compute and print the user's time-sharing usage charges for this session at the terminal and his total charges to date, and to disconnect the terminal. If the user has any temporary nonsystem files open (i.e., files created by his executed program(s)), he is given the option of saving them individually, prior to the log-off message.

Form of the Command

BYE

Usage

The user gives the BYE command when he wishes to terminate his use of the terminal. (See also the DONE command.) If the user has any temporary files open at this point, the following message is issued (prior to the log-off message):

```
n TEMPORARY FILES CREATED
```

and then each temporary file name is printed, in turn, followed by a question mark:

tempfile?

response	_meaning_
tempfile? (carriage return)	Ignore (release) this file; pass to the next file name.
tempfile? NONE	Ignore (release) this and all succeeding files; exit from system.
tempfile? SAVE _filedescr_	Save the temporary file _tempfile_ on the permanent file specified by _filename_. (_Filename_ and _tempfile_ may be synonomous.) Pass to the next file name.

See the PERM command descriptions for more detail concerning the files _tempfile_ and _filename_.

● DELETE

Function

To delete one or more complete lines from the current file, by line numbers.

Forms of the Command

(1) DELETE a

The line(s) numbered a is deleted from the current file.

(2) DELETE a, b, c, d...

The lines numbered a, b, c, d, etc., are deleted from the current file.

(3) DELETE a-b,c-d...

All lines numbered a through b, c through d, etc., are deleted from the current file.

(4) DELETE a,b,c-d,e,f-g...

The lines numbered a, b, c through d, e, and f through g, etc., are deleted from the current file.

A maximum of 12 comma-separated fields may be given.

Usage

This command is employed when the user wishes to delete some, but not all, of the lines on the current file.

If the current file was created via the OLD response or command, the original "old" permanent file is not affected by the DELETE command. (The SAVE command must be used to permanently retain any such modifications).

• DONE

Function

To cause an exit from the FORTRAN system. The system-selection question, SYSTEM ?, is reissued. Any temporary files created under FORTRAN are retained, and all accessed files remain open (file names remain in AFT).

Form of the Command

DONE

Usage

This command is employed when the user wishes to terminate his use of FORTRAN and select another system or subsystem (e.g., SCAN or ACCESS). The current file can be "carried over" to a subsequent system by responding SAME to the OLD OR NEW request.

• GET

Function

To access (i.e., open) one or more permanent files. The name of the requested file(s) is entered in the user's available file table (AFT). The user's current file (if any exist) is not affected. The GET command is designed specifically for nonquick-access files, but any type of file may be accessed with it. A number of files can be accessed with a single command.

Any nonquick-access data files to be used during execution of a FORTRAN program must be opened (entered in the user's available file table) prior to compilation/execution. Since these data files are often common to several users, the GET command simplifies the opening of such files. (See "Categories of Files," in this chapter.)

Forms of the Command

(1) GET $\underline{\text{filename}}_1$; $\underline{\text{filename}}_2$;...;$\underline{\text{filename}}_n$

Causes the quick-access type file(s) saved under the name(s) filenamei to be accessed. (A quick-access type file is one that was created with a SAVE command, by the same user.)

(2) GET $\underline{\text{filedescr}}_1$; $\underline{\text{filedescr}}_2$;...; $\underline{\text{filedescr}}_n$

Causes any permanent file(s) described by $\underline{\text{filedescr}}_n$ to be accessed, where the general form of filedescr is:

```
userID/catalog$password/...
/catalog$password/filename$password,
permission,...,permission "altname"
```

Usage

If the file to be described emanates from the user's own master catalog, the user-ID may be omitted and the file description begun with an initial slash:

```
/catalog$password/...
/filename$password,permission....
```

If the file is one of the user's own first-level files (i.e., no catalog string describing it other than the user's own user-ID), filedescr may be further shortened to simply the filename portion, with no initial slash:

```
filename$password,permission,...,
permission "altname"
```

In the simplest case, this reduces to:

```
filename$password
```

if all permissions are desired ("locking" the file out to other users), and no alternate name is desired.

A permission can be one of the following:

READ (or R)

WRITE (or W)

APEND (or A)

EXECUTE (or E)

The $password parameter must be given if a password is attached to the file. The default interpretation for permissions is READ, WRITE (i.e., if READ permission only is desired, it must be specified). Multiple "readers" of the same file are allowed by the time-sharing system.

The altname parameter, enclosed in double quotes, is given when the file filename is to be referred to by an alternate name during the current session at the terminal.

Note that if a file is accessed with a filedescr form of this command, it may then be referred to simply by its file name in further commands or in the program.

• <u>LIST</u>

<u>Function</u>

To list at the terminal all or part of a source file. Several files and/or file segments can be adjoined for listing.

<u>Forms of the Command</u>

(1) LIST

Causes the entire current file to be listed.

(2) LIST i,j (where i and j are line numbers)

Causes all lines of the current file whose line numbers lie between the values i and j, inclusive, to be listed. (Note that in the case of concatenated files -- see OLD command -- where no resequencing has been performed, multiple sets of lines numbered between i and j may or may not be listed, depending upon their sequence.)

Either the i or j parameter may be omitted; line numbers 1 and 99999999, respectively, will be assumed. That is, LIST i implies "from line i through end-of-file"; LIST, j implies "from beginning of file through line j." Parameters i and j must be positive decimal integers.

(3) LIST <u>filename</u>

Causes the entire permanent file saved under <u>filename</u> to be listed. The current file is not affected. <u>Filename</u> must include at least one alpha character.

(4) LIST <u>filename</u> (i,j)

Causes the lines numbered i through j of the permanent file saved under <u>filename</u> to be listed. The rules for i and j given under form (2) also apply here. The current file is not affected. A file with an all-numeric <u>filename</u> may be listed with this form; i.e., for the entire file: LIST <u>filename</u> ().

(5) LIST filename $(i,j)_1$; filename $(i,j)_2$;...; filename$(i,j)_n$

(Where the (i,j) line-number parameters are optional.) Causes the specified permanent files and/or file segments to be adjoined and listed, in the order given. The current file is not affected. The current file, or a segment thereof, may be specified in the file list under the name "* ".

If the list of files exceeds one line in length, it may be continued on the next line provided that the last nonblank character on the first line is a semicolon.

(6) LISTH

Causes the current file to be listed with a date-and-time heading. Forms (2) through (5) of LIST may be employed with LISTH also.

(7) LISTS a,b,c,d,...

Causes only the lines numbered a,b,c,d, etc., of the current file to be listed.

(8) LIST 99999999

Causes the highest-numbered line of the file to be printed, provided that the parameter given is either all nines or greater than the highest line number in the file.

Usage

The file(s) implied or specified by the LIST command are not altered in any way. In the cases where the current file is implied, the latest corrections or modifications are incorporated and shown.

Forms (3), (4), and (5) are for the user's own quick-access permanent files; (those files created by himself simply by means of the SAVE filename command). Files either (1) with passwords attached, or (2) emanating from a catalog string, or (3) created by another user may all be accessed first with the GET command, or accessed and listed with the filedescr forms of LIST described in Appendix A.

● <u>NEW</u>

<u>Function</u>

To enable the user to begin building a new program (i.e., new current file) from the terminal keyboard or paper tape reader at any point during his use of Time-Sharing FORTRAN. The current file is cleared of its previous contents (if any exist).

<u>Form of the Command</u>

NEW

<u>Usage</u>

If the user wishes to enter a new program and also wishes to retain the latest contents of current file (i.e., the previous "new" program or "old" program with modifications), he should employ the SAVE command prior to giving the NEW command.

- OLD

Function

Allows the user to retrieve an "old" (previously saved) program from the file system, and have a copy of it placed on the current file. Several permanent files, or portions thereof, can be concatenated on the current file.

Forms of the Command

(1) OLD filename

The file saved under filename becomes the current file.

(2) OLD filename (i,j)

Lines i through j of the file saved under filename become the current file.

Filename must be a line-numbered file.

(3) OLD $\text{filename}(i,j)_1;\text{filename}_2;\ldots;\text{filename}(i,j)_n$

(Where the (i,j) line-numbers parameters are optional.) The specified permanent files or file segments are adjoined in the order listed and become the current file. (The resulting file is not resequenced.) If the file list is too long for one line, it can be continued on the next if a semicolon is the last nonblank character on the first line before the carriage return.

The permanent file(s) specified by the OLD command are not themselves altered by any modifications subsequently made to the current file. In order to retain any such modifications, the SAVE command must be used.

Forms (1), (2), and (3) are for the user's own quick-access permanent files (those files created by himself simply by means of the SAVE filename command). Files with passwords attached, emanating from a catalog string, or created by another user may all be accessed first with the GET command, or accessed and copied with the filedescr forms of OLD described in Appendix A.

• PERM

Function

To allow a user to save, by name, a temporary file written by his FORTRAN program. This command differs from the SAVE command, in that the temporary file is specified explicitly, as well as the name of the permanent file on which it is to be saved. The saving can be done on a pre-existent (i.e., predefined) permanent file or a new permanent file can be implicitly created.

Form of the Command

PERM tempfile;filename

Causes the temporary file named by tempfile to be copied onto the permanent file filename.

Usage

When a FORTRAN program in execution attempts to write to a named file which does not already exist as a permanent file or cannot be accessed automatically (i.e., requires pre-accessing), a temporary file is created by that name. All temporary files are released when the user terminates his session at the terminal.

The temporary files that he wishes to retain may be saved either via the PERM command during his use of FORTRAN, or by means of the file-saving procedure provided by the BYE command when he is about to end his time-sharing session. If the PERM command is used, the temporary file is released immediately and the corresponding permanent file name is placed in his AFT. In this way, the saved file can be referred to by further commands. For example, the saved file can then be removed from the AFT by the way of the REMOVE command (if it is full or near full) or it can then be specified in the SCAN subsystem.

PERM works like SAVE in that if the named permanent file does not already exist, it will be created with general READ permission. To save on other than quick-access type files, the filedescr form of the PERM command, as described in Appendix A, can be employed.

● REMOVE

Function

Allows the user to delete file names from his available file table (AFT) during his session at the terminal. Since the AFT is limited in size, it may become full during a session, and unneeded files must then be removed to make room for others. Permanent files removed from the AFT are not purged from the file system. Temporary files are released.

Forms of the Command

(1) REMOVE filename

The name filename is deleted from the user's AFT.

(2) REMOVE $\underline{\text{filename}}_1$; $\underline{\text{filename}}_2$;...;$\underline{\text{filename}}_n$

The n specified filenames are deleted from the AFT.

Usage

The names of all accessed files (all files referred to by any other command except PURGE) and/or referred to by the executed object program, temporary or permanent, are entered and retained in the user's AFT. For example, if one or more permanent files (or file segments) are merely listed, the file names remain in the AFT for the duration of the user's time-sharing session, unless explicitly removed.

Since an executing FORTRAN program may access or create a number of temporary and/or intermediate files, the user's AFT could become full in mid-execution unless judicious use is made of the REMOVE command. The user can obtain a list of the files presently entered in his AFT with the command STATUS (refer to Appendix A).

• RESEQUENCE

Function

To resequence, or "clean up," the line numbers of the current file. The existent line numbers are replaced by either a standard sequence of line numbers, beginning with 0010 and incrementing by 10, or by an ordered sequence specified by the user.

Forms of the Command

(1) RESEQUENCE

Causes the complete current file to be resequenced, beginning with line number 0010 and incrementing by 10.

(2) RESEQUENCE n,m

Causes the complete current file to be resequenced, beginning with line number n, and incrementing each successive line number by m. Either the n or m parameter may be omitted.

Usage

In the debugging or modification of a source program, insertion of new lines may "clog" the original line-number sequence. If several "old" files and/or file segments are concatenated on the current file (see the OLD command), resequencing is not automatically performed and the line numbers of the resulting program are not likely to be in rational order. In either case, the RESEQUENCE command may be employed to obtain a new line-number sequence.

• RUN

Function

To compile and/or execute the implied or specified program file. Either the current file or a permanent program file can be compiled and/or executed. A concatenation of files or file segments (including all or part of the current file) may be specified as the program file. The program to be "run" may be a source or object program, or a combination of both.

The user does not specify whether or not a program or program segment is to be compiled. The RUN processor identifies source segments and compiles them automatically. The object program resulting from a compilation may be saved.

A no-go option is available to inhibit execution.

Forms of the Command

(1) RUN

The current file is compiled and executed.

(2) RUN=(NOGO)

The current file is compiled but not executed. (Useful for syntax-error checking; any errors will be noted.)

(3) RUN=<u>savefile</u>

The current file is compiled and executed, and the resultant object program is saved on the permanent file specified by <u>savefile</u>. Example: RUN=BINPROG1

(4) RUN=<u>savefile</u>(NOGO)

Combination of forms (2) and (3).

(5) RUN file-list

Where file-list has the general form:

$\underline{\text{filename}}(i,j)_1;\underline{\text{filename}}(i,j)_2;..;\underline{\text{filename}}(i,j)_n$

(The line-number parameters, (i,j), are optional.) The permanent file, or concatenation of files and/or file segments, specified by file-list is compiled and/or executed. Compilation of source segments is automatic. The current file can be included in the file list by the name "*".

Examples:

RUN SOURCE1 - Compile and execute the (source) program on file SOURCE1.

RUN BINPROG1 -- execute the (object) program on file BINPROG1.

RUN SOURCE1(10,175);* -- concatenate segment of SOURCE1 with current file (*), compile and execute.

RUN *;BINPROG2 -- concatenate current file and (object) program on BINPROG2, compile and execute.

RUN *(300,575) -- compile and execute the specified segment of the current file.

(6) RUN file-list = (NOGO)

Combination of forms (2) and (5).

(7) RUN file-list=savefile

Combination of forms (3) and (5).

(8) RUN file-list=savefile(NOGO)

Combination of forms (4) and (5). (General form)

Usage

The RUN command can be used to request a compilation and execution of a source program, a compilation only, or an execution only of an object program. Where a concatenation of files and/or file segments is specified for the program file, they may be a mixture of object and source files. (A maximum of 32 object files may appear in the file list, plus any number of source files.)

The =savefile option is the only means whereby an object program can be saved. If the file specified by savefile already exists, it must be a random file. If it does not already exist, a one-link random file will be created (for quick-access type files only). In general, savefiles should be predefined (e.g., with the ACCESS Create-File function) except in the case of small FORTRAN programs.

The filename forms of the command, shown above, are specifically for the normal, quick-access type of files. Other types of files can be specified, in file-list and savefile, with the filedescr forms described in Appendix A.

● SAVE

Function

To save the contents of the current file on a specified permanent file. The permanent file may be a pre-existent (i.e., predefined) file, or it may be created by the use of this command. The current file may be saved on more than one permanent file simultaneously.

Forms of the Command

(1) SAVE filename

The content of the current file is written onto the permanent file specified by filename. The current file is not released.

(2) SAVE filename_1;filename_2;..;filename_n

The content of the current file is written onto each of the files specified. The current file is not released.

Usage

The forms of the SAVE command given above are designed to save on, or to implicitly create, quick-access type files. These files emanate directly from the user's master catalog (identified by his user-ID), have no password attached, and have general READ permission assigned.

Files other than the quick-access type may be created and/or written to by use of other forms of SAVE, described in Appendix B; use of a combination of the GET and SAVE commands; and prior use of the ACCESS time- sharing subsystem (see Appendix B) and then the SAVE command. See "Categories of Files," below, and Appendix B for a description of nonquick-access files.

If the file specified by filename does not already exist, it will be created automatically, as defined above.

• TAPE

Function

To allow the user to build (or add to) the current file with input from the paper-tape reader of his terminal device.

Form of the Command

TAPE

Usage

The TAPE command is normally given following either a NEW response or command. (It may also be used following an OLD response or command, for substantial additions to or modification of an "old" program.) Following a TAPE command, the system responds with READY. The user must then position his tape in the reader and start the device. (See "Paper Tape Input" in this chapter for further details.)

DESCRIPTION OF FILES

File Names

The names of Time-Sharing FORTRAN files must conform to the standard time-sharing system rules as follows:

a. A file name may be from one to eight characters in length.

b. A file may be composed of any combination of alphanumerics (A-Z and 0-9), periods, and minus-signs, in any order.

(For cases where a batch-environment file with a file name longer than eight characters is to be referenced, refer to "Alternate Naming of Files," below.)

Categories of Files

In the time-sharing environment, distinctions are made between permanent files on two separate bases:

a. File-access type, which is a general time-sharing, file-system-usage distinction and is not exclusive to Time-Sharing FORTRAN; and,

b. File mode, which has primarily to do with the kinds of files produced under the Time-Sharing FORTRAN system. (Both of these categories of distinctions apply to all files.)

File-Access Types

There are three types of files, according to the method of creation and subsequent accessing of the file:

1. Quick-access files -- those permanent files that were automatically created (i.e., defined) by the system as a result of use of a SAVE filename or PERM tempfile; i.e., filename command as first reference to a particular file name. This type of file has the following characteristics:

   a. It can be accessed by its creator simply by the filename form of commands, and, in the case of data files (input or output), will be accessed automatically upon execution of a program reference to it -- i.e., it need not be pre-accessed by command.

   b. It has general READ permission assigned. It can be accessed with READ permission only by any other user who can describe it completely (creator's user-ID/filename).

2. Quick-access files with password attached -- those permanent files that (normally) were automatically created (i.e., defined) by the system as a result of use of a SAVE filename$password command as first reference to a particular file name. This type of file is the same as the simple, quick-access type described above except that it has the specified password attached. It has the following characteristics:

   It can be accessed by its creator either by the filename or the filename$password form of commands; in the former case, if $password is omitted, the system will explicitly ask for the password. Also, in the case of data files, it will be accessed automatically upon execution of a program reference to it, but the system will explicitly ask for the password. The filename$password form of the commands are described in Appendix A.

3. Nonquick-access files -- those permanent files that either do not "belong" to the user himself (i.e., were created by another user) or do not emanate directly from user's master catalog. In the latter case, the file is not completely described by user-ID and filename$password (intermediate catalogs exist), and, in general, use was made of the ACCESS subsystem in explicitly creating some or all of the catalog/file string describing the file.

The nonquick-access type of file can be accessed either with the GET command, or with similar extended forms of other commands described in Appendix A. See Appendix B for a discussion of the possible characteristics of this type of file.

Note that quick-access files (with or without password) are only quick-access type relative to the file's creator. That is to say, a quick-access file for user A is by definition not a quick-access file for any other user.

Note

> Once a type of file is initially accessed, whether by a GET or any other command, it can thenceforward be referred to simply by file name, unless explicitly removed from the AFT.

File Modes

Three modes of files may be produced under the FORTRAN system.

Mode	Characteristics
ASCII	A linked (sequential) file of variable-length records in ASCII character code; i.e., a file composed of 9-bit character strings. (An ASCII-mode file is equivalent to the standard text file.)
Binary	A linked (sequential) file of variable-length records in binary.
Random	A random file of fixed-length records in binary.

FORTRAN object programs can produce ASCII-mode files as output. ASCII is always the mode of source-program files. The file-building facilities of FORTRAN (or any other time-sharing system; e.g., EDITOR) always produce ASCII files, and only ASCII mode is acceptable on the current file.

FORTRAN object programs can produce linked binary-mode files as output.

FORTRAN object program can produce random-mode binary files as output. Random is always the mode of object-program files themselves. Saved compiler output is always in random mode.

All files, of any mode, must be explicitly saved by use of the SAVE or PERM commands in order to be retained as permanent files. If the specified permanent file does not already exist, it will be implicitly created with the correct linked or random characteristic, as required by the file mode. (Linked is the standard, or default, type of file created.)

If, however, the specified permanent file was explicitly created (predefined by the user, normally by use of the Create-File function of the ACCESS subsystem), the user must have been careful to create the file with the random (R) specification if a random-mode file is to be saved or made permanent. (See "Special Features" in Appendix B.) This is true particularly for the file specified as savefile, in the RUN statement, on which the compiler output is saved. If this is a pre-existent file, it must have previously been created -- implicitly or explicitly -- as a random file. See Appendix B for a description of the ACCESS subsystem.

Alternate Naming of Files

Permanent files may be temporarily renamed, with the altname capability of the time-sharing command language, when necessary or desirable. Alternate naming is effective only during the terminal session in which the altname is assigned and the original file name in the file system is not altered. Two cases in which alternate naming would be required are as follows:

- When a file created in the batch environment (e.g., a data file) with a name longer than eight characters is to be referred to by a Time-Sharing FORTRAN program, it must be given an alternate name of eight characters or less.

- When two or more files with identical file names are to be referred to in one time-sharing session, whether by commands or by the FORTRAN program, one or more must be differentiated by alternate names. (If the user is working only with his own quick-access files -- the "normal" case -- this problem does not arise.)

Alternate naming may conveniently be employed in the case where the file name used in a FORTRAN program and the name of the actual permanent file to be referred to do not agree. Here the file may be given an altname, rather than changing the program reference. This case might frequently arise when working with common data files.

An alternate name can be assigned with the GET command, when "pre-accessing" data (or source) files, or can be assigned with extended forms of most other commands, as described in Appendix A. Briefly, the syntax of alternate naming is:

<u>filename"altname"</u>

(or)

<u>filename$password"altname"</u>

SUPPLYING DIRECT-MODE PROGRAM INPUT

During program execution, keyboard input may need to be supplied to satisfy one or more READ statements in the user's program. Each time input is required, the equal-sign character, "=", will be printed at the terminal. The user begins typing the input immediately following the equal sign.

If the user gives a carriage return before satisfying the input-list requirements, the system will respond with another equal sign. The first carriage return following sufficient input to satisfy the list will cause program execution to resume.

EMERGENCY TERMINATION OF EXECUTION

The use of the BREAK key will terminate program execution. It will also terminate printing due to a LIST command. However, the user is cautioned against indiscriminate use of the BREAK key, since the results of its use (e.g., in regard to status of files) is completely unpredictable. Control will return to build-mode after the use of the BREAK key.

PAPER TAPE INPUT

In order to supply build-mode input from paper tape, the user gives the command TAPE. The system responds with READY. At this point, the user should position his tape in the reader and start the device. Input is terminated when either the end-of-tape occurs, the user turns off the reader, an X-OFF character is read by the paper tape reader, or a jammed tape causes a delay of over one second between the transmission of characters.

At present a maximum of 80 characters are permitted per line of paper tape input. Excessive lines will be truncated at 80 characters with the remaining data placed in the next line. A maximum of two disc links (7680 words) of paper tape input will be collected during a single input procedure. All data in excess of two disc links will be lost.

SECTION II

TIME-SHARING FORTRAN

3. Time-Sharing FORTRAN Language Characteristics

STATEMENTS

The Time-Sharing FORTRAN language may be considered a version of the FORTRAN IV language. Statements in this language are classified as follows:

- Specification statements -- provide information concerning constants and variables used in a program and information concerning storage allocation.
- Control statements -- govern flow of control in a program.
- Input/output statements -- provide necessary input/output routines and formats required.
- Arithmetic statements -- specify numerical or logical calculations.
- Subprogram statements -- enable programmer to define and use subprograms for segmentation capability.

GENERAL PROPERTIES OF FORTRAN SOURCE PROGRAMS

The order of source statements is subject to the following rules:

1. The following statements, if present, must be the first of a routine:

```
FUNCTION
SUBROUTINE
```

2. The following statements can only be preceded by the statements named in 1. If a variable in COMMON also appears in another specification statement, the COMMON statement must come first.

   COMMON
   DIMENSION
   INTEGER
   REAL
   LOGICAL
   FILENAME
   ASCII
   EXTERNAL

3. Arithmetic statement functions can only be preceded by the statements named in 1. and 2.

4. If a NAMELIST list is used in an I/O statement, it must be preceded by a NAMELIST statement definition.

5. The last statement of a program or routine must be END.

6. The statement immediately preceding the END statement of a program must be a STOP or unconditional transfer.

Variables used in COMMON statements will be automatically preset to zero.

None of the variables used in DIMENSION, DATA, or NAMELIST statements will be automatically initialized. Except for variables in COMMON statements, the user must define the values of all of his variables, including every member of an array.

Within a FORTRAN statement, spacing is not meaningful except for ASCII and filename constants, and in nH fields.

4. Constants, Variables, Subscripts, and Expressions

Time-Sharing FORTRAN provides a means of expressing quantities specified in an arithmetic formula statement through use of constants and variables.

CONSTANTS

Five types of constants are permitted:

integer	ASCII
real	filename
logical	

Integer Constants

General Form

An integer constant consists of one to 11 decimal digits written without a decimal point.

Examples:

```
3
528
8085
```

An integer constant may be as large as $2^{35}-1$, except when used for the value of a subscript or as an index of a DO, or a DO parameter, in which case the value of the integer is computed modulo 2^{18}.

Real Constants

General Form

A real constant consists of one of the following:

1. One to nine significant decimal digits written with a decimal point.

2. A sequence of decimal digits written with a decimal point, followed by a decimal exponent written as the letter E, followed by a signed or unsigned interger constant.

Examples:

21.
.203
8.0067
5.0 E3 (means 5.0 x 10^3; i.e., 5000.)
5.0 E-3 (means 5.0 x 10^{-3}; i.e., .005)
4.1E+02 (means 4.1 x 10^2; i.e., 410.)

1. The magnitude of a real constant must be between the approximate limits of 10^{38} and 10^{-38}, or must be zero.

2. A real constant has precision to approximately eight digits.

Logical Constants

General Form

A logical constant may take either of the following forms:

.TRUE.
.FALSE.

ASCII Constants

<u>**General Form**</u>

An ASCII constant consists of one to four alphameric characters, either enclosed in quotation marks or 1 to 4H followed by the literal constant. Embedded blanks are retained.

Examples:

```
"ABC"     4HKYVG
"1234"    3HDEF
"CALL"    4HNAM2
"ANS"
```

If an ASCII constant is fewer than four characters in length, it will be left-justified and filled with blanks. ASCII constants are used for storing character-coded information. An ASCII constant may be stored in a variable name or used directly, enclosed in quotes. Semicolons may not be used within an ASCII constant.

Filename Constants

<u>**General Form**</u>

A filename constant consists of one to eight alphameric characters enclosed in quotation marks or 1 to 8H followed by the literal constant. A filename constant uses two words in storage. Filename constants must be at least five characters long except when they are used in input/output statements; then they may be shorter than five characters. Embedded blanks are retained.

Examples:

```
"ABCDE"       6HCASE1Ø
"SAM"         8HPROTECTS
"FILE 128"    5HCODED
"12381632"
```

If a filename constant consists of fewer than eight characters, it will be left-justified and filled with blanks to complete the two words. Thus "SAM" is stored as SAMbbbbb. A filename constant may be stored in a variable name or used directly, enclosed in quotes.

VARIABLES

The mode of a variable is specified by its name or by a Type statement. There are six types of variables:

integer	filename
real	ASCII
logical	external

Variable Names

<u>General Form</u>

A variable name consists of alphameric characters, the first of which must be alphabetic.

Examples:

```
L5
JOB1
BETATS
COST
K
```

A variable name may be any length, but when part of a NAME LIST input/output list or when it appears in an EXTERNAL statement, only the first eight characters are considered significant.

Variable Type Specification

The type of a real variable or function name and an integer variable or function name may be specified in one of two ways: implicitly by name, or explicitly by a Type statement. (See the sections "Type Statements" in Chapter 5 and "Naming Subroutines" in Chapter 9.) All other variables must have their type specified by a Type statement. These include ASCII, filename, logical, and external variables.

Implicit Type Assignment

Implicit type assignment pertains only to real and integer variable and function names:

1. If the first character of the name is I,J,K,L,M, or N, it is an integer name; e.g., MAX, JOB, IDIST, LESL.

2. If the first character of the name is not I,J,K,L,M, or N, it is a real name; e.g., ALPHA, BMAX, Q, WHIT.

ASCII and Filename Variables

ASCII and filename variables consist of four and eight characters respectively of character-coded alphanumeric information.

Filename specifies that the variables that follow are either the names of files or contain some type of alpha information up to eight characters long. If less than eight characters are used in the information, they are left-justified and the remainder filled with blanks. Filename variables may not be dimensioned, so no more than eight characters will be accepted. For more than eight characters, dimensioned ASCII should be used.

ASCII is used to type character-coded information formerly classified as hollerith. Each variable name is dimensioned according to the number of four-character words it uses. If less than four characters are used, they are left-justified and the remainder filled with blanks.

Examples: (These variables must first be typed as ASCII or filename prior to their appearance elsewhere in the program.)

ASCII	FILENAME
ARRAY2(8)	PROJECTX
BETA	AVGAMT
TITLE(4)	XY
SIZE(3)	CASES
ALPHA	COM

External Variables

An external variable is the name of a subprogram that appears in the calling sequence to another subprogram or built-in function used as the name of a FUNCTION or SUBROUTINE subprogram. An external variable may be any length, but only the first eight characters are checked by the system for identification. It must appear in an EXTERNAL Type statement at the beginning of the source program.

SUBSCRIPTS

A variable may be made to represent any element of an array containing from one to 63 dimensions by appending one, two, three,..., or 63 subscripts, respectively, to the variable name. The variable is then a subscripted variable. The value of the subscripts determines the member of the array to which reference is made.

Form of Subscripts

<u>General Form</u>

A subscript may take the form of any legal FORTRAN arithmetic expression.

Examples:

```
IMAS     8*IQUAN    9+J
J9       5*L+7      B**2
K2       4*M-3      6**A-(1-SQRT(3.14))/8
N+3      7+2*K
```

Note: The value of a subscript expression must be greater than zero and not greater than the corresponding array dimension. The value of a subscript expression containing real variables is truncated to an integer after evaluation.

Subscripted Variables

<u>General Form</u>

A subscripted variable consists of a variable name, followed by parentheses, enclosing one to 63 subscripts separated by commas.

Examples:

```
A(I)
K(3)
BETA (8*J+2, K-2, L)
MAX (K,J,K,L,M,N)
```

1. During the execution, the subscript is evaluated so that the subscripted variable refers to a specific member of the array.

2. Each variable that appears in subscripted form must have the size of the array specified. This must be done by a DIMENSION statement or by a COMMON or Type statement (except EXTERNAL) that contains dimension information.

Arrangement of Arrays in Storage

Arrays are stored in column order in increasing storage locations, with the first of their subscripts varying most rapidly and the last varying least rapidly.

For example:

The 2-dimensional array A(m,n) is stored as follows, from the lowest core storage location to the highest:

$$A_{1,1}, A_{2,1}, \ldots, A_{m,1}, A_{1,2}, A_{2,2}, \ldots,$$
$$A_{m,2}, \ldots, A_{m,n}$$

EXPRESSIONS

The FORTRAN language includes two kinds of expressions:

arithmetic
logical

Arithmetic Expressions

An arithmetic expression consists of certain sequences of constants, subscripted and nonsubscripted variables, and arithmetic function references separated by arithmetic operation symbols, commas, and parentheses.

The following arithmetic operation symbols denote addition, subtraction, multiplication, division, and exponentiation, respectively:

```
+
-
*
/
**
```

The following are the rules for constructing arithmetic expressions:

1. Any expression may be enclosed in parentheses.

2. Expressions may be connected by the arithmetic operation symbols to form other expressions, provided that:

   a. No two operators appear in sequence.

   b. No operation symbol is assumed to be present.

   For example:

   (X)(Y) is invalid.

3. Preceding an expression by a plus or minus sign does not affect the type of the expression.

4. In the hierarchy of operations, parentheses may be used in arithmetic expressions to specify the order in which operations are to be computed. Where parentheses are omitted, the order is understood to be as follows (from innermost operations to outermost operations):

   a. Function Reference

   b. **(exponentiation)

   c. * and / (multiplication and division)

   d. + and - (addition and subtraction)

   Expressions are evaluated from left to right where operators are on the same level except for exponentiation, where they are evaluated from right to left.

Logical Expressions

A logical expression consists of certain sequences of logical constants, logical variables, references to logical functions, and arithmetic expressions separated by logical operation symbols or relational operation symbols. A logical expression always has the value:

.TRUE. or .FALSE.

The logical operation symbols (where a and b are logical expressions) are:

Symbol	Definition
.NOT.a	This has the value .TRUE. only if a is .FALSE.; it has the value .FALSE. only if a is .TRUE.
a.AND.b	This has the value .TRUE. only if a and b are both .TRUE.; otherwise it is .FALSE.
a.OR.b	(Inclusive OR) This has the value .TRUE. if either a or b is .TRUE.; it has the value .FALSE. only if both a and b are .FALSE.

The logical operators NOT, AND, and OR must always be preceded and followed by a period.

The relational operation symbols are:

Symbol	Definition
.GT.	greater than
.GE.	greater than or equal to
.LT.	less than
.LE.	less than or equal to
.EQ.	equal to
.NE.	not equal to

The relational operators must always be preceded and followed by a period. These are the only operation symbols that work with ASCII. Filename variables or constants cannot be used in logical expressions.

The logical expression will have the value .TRUE. if the condition expressed by the relational operator is met; otherwise, the logical expression will have the value .FALSE.. Rules for constructing logical expressions are:

1. A logical expression may consist of a single logical constant, a logical variable, or a reference to a logical function.

2. The logical operator .NOT. must be followed by a logical expression, and the logical operators .AND. and .OR. must be preceded and followed by logical expressions to form more complex logical expressions.

3. Any logical expression may be enclosed in parentheses; however, the logical expression to which the .NOT. applies must be enclosed in parentheses if it contains two or more quantities.

4. In the hierarchy of operations, parentheses may be used in logical expressions to specify the order in which operations are to be computed. Where parentheses are omitted, the order is understood to be as follows (from innermost to outermost operation):

   a. Function Reference
   b. ** (exponentiation)
   c. * and / (multiplication and division)
   d. + and - (addition and subtraction)
   e. .LT.,.LE.,.EQ.,.NE.,.GT.,.GE.
   f. .NOT.
   g. .AND.
   h. .OR.

Examples:

```
.NOT. (A.EQ.B)
C .AND. D
(A.GT.B) .OR. (F.LT.E)
(L.EQ.M) .AND. (Q.NE.P)
K.OR. (W.LE.V)
.NOT. (M.AND.N)
```

5. Specification Statements

Specification statements are nonexecutable statements which provide the compiler with information about storage allocation and about the constants and variables used in the program.

DIMENSION STATEMENT

General Form

DIMENSION $v_1(i_1),v_2(i_2),...$

where:

1. Each v_n is an array variable.
2. Each i_n is composed of from one to 63 unsigned integer constants and/or integer variables, separated by commas. (Integer variables may be components of i_n only when the DIMENSION statement appears in a FUNCTION or SUBROUTINE subprogram).

Examples:

```
DIMENSION A(1,2,3,4), B(10)

DIMENSION C (2,2,3,3,4,4,5)
```

In the preceding examples, A,B, and C are declared to be array variables with four, one, and seven dimensions, respectively.

The DIMENSION statement provides the information necessary to allocate storage for arrays in the object program. The DIMENSION statement defines the maximum size of arrays. An array may be declared to have from one to 63 dimensions by placing it in a DIMENSION statement with the appropriate number of subscripts appended to the variable.

1. The DIMENSION statement must precede every statement in the program and immediately follow the subprogram defining statement in the case of a FUNCTION or SUBROUTINE subprogram. For exceptions, see Chapter 3.

2. A single DIMENSION statement may specify the dimensions of many arrays.

3. If a variable appears in a DIMENSION statement, it must not be dimensioned elsewhere.

4. Dimensions may also be declared in a COMMON or a Type statement. If this is done, then these statements are subject to the rules for the DIMENSION statement.

Adjustable Dimensions

The name of an array and the constants that are its dimensions may be passed as arguments in a subprogram call. In this way, a subprogram may perform calculations on arrays whose sizes are not determined until the subprogram is called. Figure 1 illustrates the use of adjustable dimensions.

1. Variables may be used as dimensions of an array only in the DIMENSION statement of a FUNCTION or SUBROUTINE subprogram. For any such array, the array name and all the variables used as dimensions must appear as arguments in the FUNCTION or SUBROUTINE statement.

```
SUBROUTINE MAMY(...,N,L,M,...)
DIMENSION...,N(L,M),...
          .
          .
          .
DO 100 I=1,L
```

Figure 1

2. The adjustable dimensions may not be altered within the subprogram.

3. The absolute dimensions must be specified in a DIMENSION statement of the calling program.

4. The calling program passes the specific dimensions to the subprogram. These specific dimensions are those that appear in the DIMENSION statement of the calling program. Variable dimension size may be passed through more than one level of subprogram.

5. A FUNCTION or SUBROUTINE argument must be explicitly declared as INTEGER, unless it is implicitly of integer type.

Example:

```
      DIMENSION K(4,5), J(2,3), C(10)
           .
           .
           .
      CALL SETFLG (K,J,4,5,2,3)
           .
           .
           .
      SUBROUTINE SETFLG (K,J,I,L,M,N)
      DIMENSION K(I,L),J(M,N)
           .
           .
           .
      DO 20 NO=1,I
      DO 20 MO=1,L
      K(NO,M))=0
20    CONTINUE
           .
           .
           .
```

COMMON STATEMENT

<u>General Form</u>

COMMON a,b,c,...

where a,b,c,... are variables that may be dimensioned.

Examples:

```
COMMON A,B,K,LEP,VW,P
COMMON D(5,10,12) ,EF,N(20) ,XY(2,6) ,HT
```

Variables, including array names, appearing in a COMMON statement may be shared by a program and its subprograms.

1. The COMMON statement may only be preceded by FUNCTION or SUBROUTINE statements. (For details, see Chapter 3.) If the variables appearing in a COMMON statement contain dimension information, they must not be dimensioned elsewhere.

2. The locations in the COMMON area are assigned in the sequence in which the variables appear in the COMMON statement, beginning with the first COMMON statement of the program.

TYPE STATEMENTS

The type of a variable or function may be specified by means of one of the six Type statements.

<u>General Form</u>

```
INTEGER a(i1),b(i2),c(i3),...
REAL a(i1),b(i2),c(i3),...
LOGICAL a(i1),b(i2),c(i3),...
EXTERNAL x,y,z,...
FILENAME d,e,f,...
ASCII g(i1),h(i2),j,...
```

where:

1. a,b,c,... are variable or function names appearing within the program.
2. x,y,z,... are subprogram names appearing within the program.
3. Each i_n is composed of from one to 63 integer constants and/or integer variables. Subscripts may be appended only to variable names appearing within the program, not to function names.
4. Filename variables are the names of files used in the program.
5. ASCII variables are dimensioned according to the number of 4-character words they use.

Examples:

```
INTEGER   BIXF,X,QF,LSL
REAL      IMIN,LOG,GRN,KLW
LOGICAL   A(10,10),B
EXTERNAL  SIN,MATMPY,INVTRY
FILENAME  AA,FG,VEC,POSITION
ASCII     NAME(6),RAD(7)
```

The variable or function names following the type (INTEGER, REAL, etc.) in the Type statement are defined to be of that type, and remain so throughout the program; the type may not be changed. Note that LSL and GRN need not appear in their respective Type statements since their type is implied by their first characters.

1. The appearance of a name in any Type statement, except EXTERNAL, overrides the implicit type assignment.

2. Variables that appear in EXTERNAL statements are subprogram names. Subprogram names must appear in an EXTERNAL statement if they are the arguments of other subprograms or if they are the name of a built-in function that is used as the name of a FUNCTION or SUBROUTINE subprogram.

3. A name may appear in two Type statements only if one of the statements is EXTERNAL.

4. The Type statements (except EXTERNAL) must precede the first appearance of the variable(s) to which they refer in any executable, NAMELIST, or DATA statements in the program.

5. The EXTERNAL statement may not be used to dimension variables.

6. Any variable that is dimensioned by a Type statement may not be dimensioned elsewhere; i.e., it may not appear as a dimensioned variable in a COMMON or DIMENSION statement.

7. A name declared to be of a given type may assume only the values of a constant of the same type.

DATA STATEMENT

Data may be compiled into the object program by means of the DATA statement.

<u>General Form</u>

DATA list/d_1,d_2,...,d_n/,list/d_1,d_2,k*d_3,...,d_m/,...

where:

1. List contains the names of the variables being defined.

2. d is the data literal.

3. k is an integer constant.

Examples:

```
DATA R,Q/14.2,3HEND/,Z/0777777711111/
DATA B(2),B(3),W,T/2.0,3.0,"STAR","TIME"/
LOGICAL LA,LB
DATA LA,LB/.TRUE.,.FALSE./
```

1. <u>List</u>. Subscripted variables may appear in the list. Where a subscript symbol is used, it must be an integer constant.

2. <u>k.</u> The letter k denotes an integer constant and may appear before a d-field to indicate that the field is to be repeated k times. An asterisk must follow the constant k to separate it from the field to be repeated.

3. The data literals may take any of the four following forms:

   (a) Integer and real constants. They may be signed or unsigned.

   (b) Alphameric characters. The alphameric field is written as nH followed by n alphameric characters, or the characters may be enclosed in quotes. Each group of four or fewer alphameric characters forms a word. n may be less than or equal to four, if the word is an ASCII constant, or n may be five to eight, if the word is a filename constant.

   (c) Octal digits. The octal field is written as the letter O, followed by one to 12 unsigned octal digits.

   (d) Logical constants. The logical field may be written as either .TRUE. or .FALSE..

4. There must be a one-to-one correspondence between the list items and the data literals. Each data literal (integer constant, real constant, alphameric constant, logical constant, or octal constant) corresponds to one undimensioned variable or subscripted array reference.

5. DATA defined variables that are redefined during execution will assume their new values regardless of the DATA statement.

6. Where data is to be compiled into an entire array, the name of the array (with indexing information omitted) can be placed in the list. The number of data literals must be equal to the size of the array.

   For example, the statements

   ```
   DIMENSION B(25)
   DATA A,B,C,/24*4.0,3.0,2.0,1.0/
   ```

   define the values of A,B(1),...,B(23) to be 4.0 and the values of B(24), B(25), and C to be 3.0, 2.0 and 1.0, respectively.

7. The DATA statement may not be used to enter data into COMMON.

8. Implied DO loops may not be used in the list items to define an array.

6. The Arithmetic Statement

The arithmetic statement defines a numerical or logical calculation. It consists of a variable name followed by an equal sign, followed in turn by any desired expression. The equal sign of the FORTRAN statement specifies replacement; the expression may be single constant, a single variable, or a complex combination of operations. In essence, the machine computes the complete expression on the right of the equal sign and assigns that computed value to the variable whose name appears on the left of the equal sign. The form of the statement is limited because only a single variable may appear on the left of the equal sign.

General Form

a=b

where:

1. a is a subscripted or nonsubscripted variable.
2. b is an expression.

Examples:

```
Q1=K                       JOE="SAM"
A(1)=B(1)+SIN(C(1))        IARRAY(2)="FEB"
V=.TRUE.                   R="CALLTYP"
E=C.GT.D.AND.F.LE.G        BEC="ABCDE"
```

Figure 2 indicates which type expressions may be equated to which type of variable in an arithmetic statement. In Figure 2, Y indicates a valid statement and N indicates an invalid statement.

Right Side Of Equal Sign

Left Side Of Equal Sign	Expression / Variable	Real	Integer	Logical	ASCII	Filename
	Real	Y	Y	N	N	N
	Integer	Y	Y	N	N	N
	Logical	N	N	Y	N	N
	ASCII	N	N	N	Y	Y
	Filename	N	N	N	Y	Y

Figure 2

Examples:

A=B	Replace A by the current value of B.
I=B	Truncate B to an integer, convert it to an integer constant, and store it in I.
A=I	Convert I to a real variable and store it in A.
I=I+1	Add 1 to I and store it in I.
A=3*B	This expression is mixed; it contains both a real variable and an integer constant. Multiply B by 3 and store it in A.

LOGICAL G,H
G=.TRUE.
H=.NOT.G
	Store the logical constant .TRUE. in G. IF G is .TRUE., store the value .FALSE. in H; if G is .FALSE. store the value .TRUE. in H.

LOGICAL H H=I.GE.A	H is .TRUE. If I is greater than or equal to A; H is .FALSE. otherwise.
FILENAME JOE JOE="PETE"	PETE, an ASCII constant, is stored in JOE.
FILENAME BEC BEC="ACCOUNTS"	ACCOUNTS, a filename constant, is stored in BEC.
ASCII R R="CALLTYP"	CALLTYP, a filename constant, is stored in R as "CALL".
ASCII IARRAY (12) IARRAY (2)="FEB"	FEB, an ASCII constant, is stored in IARRAY (2).
LOGICAL G,H,P G=H.OR.(.NOT. P)	Two logical operators may appear in sequence only if the second one is .NOT..

1	2	3	4
H	P	~P	Hv ~ P
T	T	F	T
T	F	T	T
F	T	F	F
F	F	T	T

In the above figure, ~ implies .NOT. and v implies .OR.. The chart is interpreted as follows:

If column 1 and column 2 hold, then column 3 and column 4.

LOGICAL G G=3..GT.B	G is .TRUE. if 3. is greater than B; G is .FALSE. otherwise.

The last two examples illustrate the following rule. Two decimal points may appear in succession only if 1) two logical operators appear in sequence where the second one must be .NOT. or 2) a constant with a decimal point precedes or follows a relational operator.

7. Control Statements

Control statements enable the programmer to control and terminate the flow of his program.

UNCONDITIONAL GO TO STATEMENT

General Form

```
GO TO n
```

where:

n is a statement number.

Example:

```
        GO TO 25
```

This statement causes control to be transferred to the statement numbered n.

COMPUTED GO TO STATEMENT

General Form

GO TO (n_1,n_2,...,n_m),i

where:

1. n_1,n_2,...,n_m are statement numbers.
2. i is a nonsubscripted integer variable.

Example:

```
          GO TO (30,45,50,9),K
```

This statement causes control to be transferred to the statement numbered n_1,n_2,...n_m depending on whether the value of i is 1,2,3,...,m respectively, at the time of execution. Thus, in the example, if K is 3 at the time of execution, a transfer to the third statement number in the list (i.e., statement 50) will occur. If the value of K is greater than m or if K=0, the user will be given a message and aborted.

ASSIGNED GO TO STATEMENT

<u>General Form</u>

GO TO i, ($n_1, n_2, \ldots, n_m$)

where:

1. i is a nonsubscripted integer variable appearing in a previously executed ASSIGN statement.
2. $n_1, n_2, \ldots, n_m$ are statement numbers.

Example:

```
        GO TO J, (17,12,19)
```

This statement causes control to be transferred to the statement number last assigned to i by an ASSIGN statement; $n_1, n_2, \ldots, n_m$ is a list of the m values that i may assume.

ASSIGN STATEMENT

General Form

ASSIGN n TO i

where:

1. n is a statement number.

2. i is a nonsubscripted integer variable that appears in an assigned GO TO statement.

Examples:

```
ASSIGN 12 to K
ASSIGN 37 to JA
```

This statement causes a subsequent GO TO i, $(n_1, n_2, \ldots n_m)$ to transfer control to the statement numbered n, where n is one of the statement numbers included in the series $n_1, n_2, \ldots, n_m$.

ARITHMETIC IF STATEMENT

General Form

IF (a) n_1, n_2, n_3

where:

1. a is an arithmetic expression.
2. n_1,n_2,n_3 are statement numbers.

Examples:

```
        IF (A(J,K)-B) 10,4,30
        IF (D*E+BRN) 9,9,15
```

This statement causes control to be transferred to the statement numbered n_1, n_2, or n_3 if the value of a is less than, equal to, or greater than zero, respectively. n_3 must always be included in the statement.

LOGICAL IF STATEMENT

<u>General Form</u>

IF (t) s

where:

1. t is a logical expression.

2. s is any executable statement except DO or another logical IF.

Examples: In these examples A,B,D, and Q are logical variables.

```
IF (A.AND.B)  F=SIN (R)
IF (16.GT.L) GO TO 24
IF (D.OR.X.LE.Y) GO TO (18,20),I
IF (Q) CALL SUB
```

1. If the logical expression t is true, statement s is executed. Control is then transferred to the next sequential statement unless s is a transfer statement, in which case, control is transferred as indicated.

2. If t is false, control is transferred to the next sequential statement.

3. If t is true and s is a CALL statement, control is transferred to the next sequential statement upon return from the subprogram.

DO STATEMENT

General Form

DO n i= m_1, m_2, m_3

where:

1. n is a statement number which defines the range of the DO.

2. i is a nonsubscripted integer variable, called the index.

3. m_1, m_2, m_3 are each either an unsigned integer constant or a nonsubscripted integer variable; if m_3 is not stated, it is taken to be 1.

Examples:

```
     DO 30 I=1,M,2
     DO 24 I=1,10
```

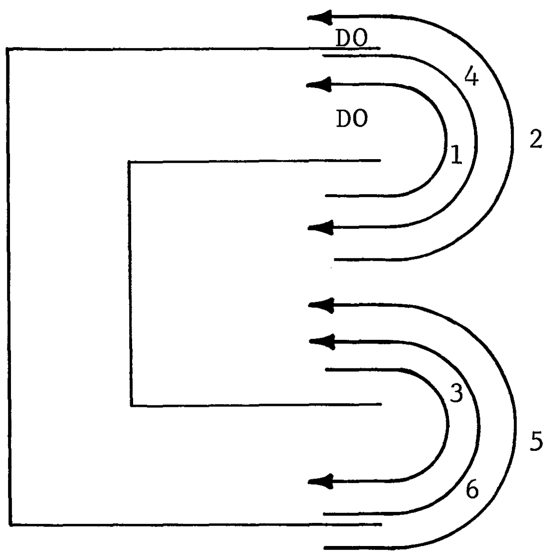
The DO statement is a command to execute repeatedly the statements that follow, up to and including the statement numbered n. The statements in the range of the DO are executed repeatedly with i equal to m_1, then i equal to m_1+m_3; then i equal to m_1+2m_3, etc., until i is equal to the highest value in this sequence that does not exceed m_2. Regardless of the initial values of m_1, m_2, and m_3, the statements in the range of the DO will be executed at least once. The value of m_1, m_2, and m_3 must be greater than zero when the DO statement is executed.

1. The <u>range</u> of a DO is that set of statements that will be executed repeatedly; i.e., it is the sequence of consecutive statements immediately following the DO statement, up to and including the statement numbered n. After the last execution of the range, the DO is said to be satisfied.

2. The index of a DO is the integer variable i. Throughout the range of the DO, the index is available for computation, either as an ordinary integer variable or as the variable of a subscript. Upon exiting from a DO by satisfying the DO, the index i must be redefined before it is used in computation. Upon exiting from a DO by transferring out of the range of the DO, the index i is available for computation and is equal to the last value it attained.

3. Within the range of a DO statement may be other DO statements; such a configuration is called a DO nest. If the range of a DO includes another DO, then all of the statements in the range of the latter must also be in the range of the former.

4. Transfer of Control and DO Statements. Control may not be transferred into the range of a DO from outside its range. Thus, in the configuration following, 1,2, and 3 are permitted transfers, but 4,5, and 6 are not.



5. Restrictions on Statements in the Range of a DO.

   (a) Any statement that redefines the index or any of the indexing parameters is not permitted in the range of a DO.

   (b) The range of a DO cannot end with an arithmetic IF or GO TO-type statement, with a nonexecutable statement, or with a RETURN or STOP statement. The range of a DO may end with a logical IF, in which case, control is handled as follows:

      IF (t)s
      If the logical expression t is false, the DO is reiterated; if the logical expression t is true, statement s is executed and then the DO is reiterated.

   However, if t is true and s is an arithmetic IF or transfer type statement, control is transferred as indicated.

6. When a reference to a subprogram is executed in the range of a DO, care must be taken that the called subprogram does not alter the DO index or the indexing parameters.

CONTINUE STATEMENT

General Form

```
CONTINUE
```

CONTINUE is a dummy statement that gives rise to no instructions in the object program. It is most frequently used as the last statement in the range of a DO to provide a transfer address for IF and GO TO statements that are intended to begin another repetition of the DO range.

END STATEMENT

General Form

END

1. The END statement terminates compilation of a program.
2. The END statement must be the last statement of the program.

STOP STATEMENT

General Form

STOP

The STOP statement terminates the execution of any program by returning control to the system.

8. Input/Output Statements

The FORTRAN statements that specify transmission of information to or from input/output devices may be grouped as follows:

- General Input/Output Statements

  The statements READ, PRINT, and WRITE cause the transmission of a specified list of quantities between the program and an input/output device.

- Manipulative Input/Output Statements

  The statements CLOSE FILE, BEGIN FILE, END FILE, and BACKSPACE manipulate files.

- Nonexecutable Statements

  Either of two nonexecutable statements (the FORMAT statement or the NAMELIST statement) may be used with the general input/output statements.

  The FORMAT statement, which can be used with any general input/output statement, specifies the arrangement of data in the external input/output medium. If the FORMAT statement is referred to by a READ statement, the input data must meet the specifications described in "Data Input Referring to a FORMAT Statement" in this chapter.

  The NAMELIST statement specifies an input/output list of variables and arrays. Input/output of the values associated with the list is effected by reference to the list in a READ or WRITE statement. If the NAMELIST statement is referred to by a READ statement, the input data must meet the specifications described in "Data Input Referring to a NAMELIST Statement" in this chapter.

LIST SPECIFICATIONS

When arrays or variables are transmitted, an ordered list of the quantities to be transmitted must be included either in the general input/output statement or the referenced NAMELIST. The order of the input/output list must be the same as the order in which the information exists in the input/output medium.

1. An input/output list is a string of list items separated by commas. A list item may be:

   a. An expression (on output only)
   b. An implied DO
   c. An array name

   An input/output list reads from left to right with repetition of variables enclosed in parentheses.

   Examples: A, B, C*D**E,1.2,Z,SQRT (14.6),
   D, G, (B (K), K = 1,4), COS (1.22), 50

   Consider the following input/output list:

```
  A,B(3), (C(I), D(I,K), I = 1, 10),
((E(I,J), I = 1, 10, 2),F(J,3),J = 1,K)
```

   This list implies that the information in the external input/output medium is arranged as follows:

```
A,B(3), C(1), D(1,K), C(2), D(2,K),...,
C(10), D(10,K), E(1,1), E(3,1),...,
E(9,1), F(1,3), E(1,2), E(3,2),....,
E(9,2), F(2,3),..., F(K,3)
```

2. The execution of an input/output list is exactly that of a DO loop, as though each left parenthesis (except subscripting parentheses) were a DO, with indexing given immediately before the matching right parenthesis, and with the DO range extending up to that indexing information. The order of the input/output list above may be considered equivalent to the following program statements:

```
       A
       B(3)
       DO 5 I=1, 10         }
       C (I)                } (C(I), D(I,K), I=1, 10)
   5   D(I,K)               }
       DO 9 J=1, K          }
       DO 8 I=1, 10, 2      } ((E(I,J), I=1, 10,2),
   8   E(I,J)               } F(J,3) , J=1,K)
   9   F(J,3)               }
```

3. An implied DO is best defined by an example. In the input/output list above, the list item (C(I), D(I,K),I=1, 10) is an implied DO; it is evaluated as in the above program.

   The range of an implied DO must be clearly defined by parentheses.

4. For a list of the form K, A(K), or K (A(I), I=1,K), where the definition of an index or an indexing parameter appears earlier in the list of an input statement than its use, the indexing will be carried out with the newly read-in value.

5. Any number of quantities may appear in a single list. Essentially, it is the list that controls the quantity of data read. If more quantities are to be transmitted than are in the list, only the number of quantities specified in the list are transmitted, and remaining quantities are ignored. Conversely, if a list contains more quantities than are given in one ASCII record, more records are read or blanks are supplied depending upon the FORMAT statement; if a list contains more quantities than are given in nonrandom binary record, reading is terminated as an object program error.

6. By specifying an array name in the list of an input/output statement or a NAMELIST, an entire array can be designated for transmission between core storage and an input/output medium. Only the name of the array need be given and the indexing information may be omitted.

   Example:

   ```
   DIMENSION A (5,5)
   .
   .
   READ:A
   ```

   In the above example, the READ statement shown is sufficient to read in the entire array; the array is stored in column order in increasing storage locations, with the first subscript varying most rapidly, and the last varying least rapidly.

FREE-FIELD ASCII INPUT/OUTPUT STATEMENTS

The following input/output statements enable a user to transmit a list of quantities without reference to a FORMAT or NAMELIST statement. The type of each variable in the list determines the conversion to be used.

<u>**General Form**</u>

```
     READ:list
     PRINT:list
     READ(f,n1,END=n2)list
n1   FORMAT(V)
     WRITE(f,n1)list
n1   FORMAT(V)
```

The READ:list statement causes the character = to be sent to the user's teletype as a signal to input a line of data. Successive = signs are sent until the list has been satisfied. List items are delimited by a comma, carriage return (or EOM), or blanks in the absence of the other delimiters. Leading and trailing blanks are ignored. Each line of data is terminated by a carriage return (or EOM). Data lines ended with a comma followed by a carriage return will result in a null field being supplied for the list item following the one whose input was terminated by the comma.

The PRINT:list causes a printed line of ASCII output preceded by a carriage return and line feed to be transmitted to the teletype. Each printed line is 72 or fewer characters in length, and the conversion formats used are shown in the table below. The list may include alphameric information enclosed in quotes in addition to the other forms of list items.

The READ(f,n_1,END=n_2)list statement causes the elements of the list to be read from file f according to the conversion formats shown in the table. n_1 is a format statement number for FORMAT (V) where V is constant and means a free-field format. n_2 is a statement number to which transfer is made upon reaching an end-of-file. Normally, FORMAT (V) is used to read files created with FORMAT (V).

The WRITE(f,n_1)list statement causes the elements of the list to be written on file f according to the conversion formats shown in the table. n_1 is a format statement number for FORMAT (V) where V is constant and means a free-field format.

Examples:

```
READ:(A(I), I=1,5), B, J,X
```

When an = sign is transmitted to the teletype, the user might input the following data to satisfy the list:

```
=3.22,4.1,5.3,67,80,1.14,51.6,12,1.7

PRINT:D, (L(K),K=1,2), "ANSWER","DATE"
```

The following line of output might result from this print statement:

```
        2.7281500E+02     205617     8134ANSWERDATE

        READ ("FILE1",10,END=55)SAM,MAP,B,C,P

     10 FORMAT (V)

        WRITE ("FILE1",35)SLP,TEK,L,M,N,J

     35 FORMAT (V)
```

In the table below, w represents the number of characters in the blank or comma-separated strings.

FREE-FIELD CONVERSION FORMATS

Type of Variable	READ:	PRINT:
real	E(or F)w.d	1PE16.7
integer	Iw	I12
logical	Lw	L2
filename	Aw	A9
ASCII	Aw	A5

FORMATTED ASCII INPUT/OUTPUT STATEMENTS

The formatted ASCII input/output statements enable a user to transmit a list of quantities by referencing a FORMAT statement that describes the type of conversion to be performed between the internal machine language and the external notation for each quantity in the list. The forms of these statements are given in the following table, where f is a file name, n_1 is a FORMAT statement number, and n_2 is a statement number to which transfer is made upon reaching an end of file.

General Form	Type of Input/Output
READ n_1, list	ASCII terminal input
READ (f, n_1, END=n_2)list	ASCII file input
PRINT n_1, list	ASCII terminal output
WRITE (f,n_1)list	ASCII file output

Examples:

```
READ 10, (C(I), I=1,5)

READ ("DICK",10)A,B,(D(J), J=1,10)

PRINT 20, A, (C(K), K=1,5), TAN(.83)

WRITE ("FILE", 10) D,G, (H(I), I=1,4),M**2
```

The READ n, list statement causes the character = to be sent to the user's teletype as a signal to input a line of data. Input data is converted according to the format specified in statement n. Successive = signs are sent as determined by the format statement until the list has been satisfied.

The READ(f,n) list statement causes ASCII information to be read from file f according to the format specified in statement n. The use of this statement with a null file name is equivalent to READ n, list. A null file name is one consisting of all blanks.

The PRINT n, list statement causes ASCII information to be transmitted to the user's teletype according to the format specified in statement n.

The WRITE(f,n) list statement causes ASCII information to be transmitted to file f according to the format specified in statement n. The use of this statement with a null file name is equivalent to PRINT n, list.

The first character of each record supplied for the PRINT n, list statement is considered a carriage-control character and is not printed.

FORMAT STATEMENT

The formatted ASCII input/output statements require, in addition to a list of quantities to be transmitted, reference to a FORMAT statement that describes the type of conversion to be performed between the internal machine language and the external notation for each quantity in the list.

<u>General Form</u>

FORMAT $(S_1,S_2,\ldots,S_n/S'_1,S'_2,\ldots,S'_n/\ldots)$

where

each field, S_i, is a format specification.

Example:

```
          FORMAT (I2/(E12.4,F10.2))
```

1. FORMAT statements are not executed; they may be placed anywhere in the source program. Each FORMAT statement must be given a statement number.

2. The FORMAT statement indicates, among other things, the maximum size of each record to be transmitted. In this connection, it must be remembered that the FORMAT statement is used in conjunction with the list of some particular input/output statement, except when a FORMAT statement consists entirely of alphameric fields. In all other cases, control in the object program switches back and forth between the list, which specifies whether data remains to be transmitted, and the FORMAT statement, which gives the specifications for transmission of that data.

3. Slashes are used to specify unit records, which must be one of the following.

   a. A file record
   b. A line to be read from or printed on the teletype

Thus, FORMAT (3F9.2,2F10.4/8E14.5) would specify records in which the first, third, fifth, etc., have the format (3F9.2,2F10.4), and the second, fourth, sixth, etc., have the format 8E14.5).

4. During input/output of data, the object program scans the FORMAT statement to which the relevant input/output statement refers. When a specification for a numerical field is found and list items remain to be transmitted, input/output takes place according to the specifications, and scanning of the FORMAT statement resumes.

If no items remain, transmission ceases and execution of that particular input/output statement is terminated.

Numeric Fields

Five types of conversion are available for numeric data:

Internal	Conversion Code	External
Floating point	E	Real with E exponent
Floating point	F	Real without exponent
Any	G	Appropriate type
Integer	I	Decimal Integer
Integer	O	Octal Integer

These types of conversion are specified, in the forms Ew.d, Fw.d, Gw.d, Iw, Ow, where:

1. E, F, G, I, and O represent the type of conversion.

2. w is an unsigned integer constant that represents the field width for converted data; this field width may be greater than required to provide spacing between numbers.

3. d is an unsigned integer or zero that represents the number of positions of the field that appear to the right of the decimal point. For E-, F-, and G-conversion, d will be made equal to eight if it should be equal to nine.

For example, the statement FORMAT (I2, E12.4, O8, F10.4) might cause the following line to be printed:

```
27b̸-0.9321Eb̸025773427b̸b̸b̸-0.0076
```

where b̸ indicates a blank space.

The following are notes on E-, F-, G-, I-, and O-conversion.

1. Specifications for successive fields are separated by commas and/or slashes. (See "Multiple-Record Formats" in this chapter.)

2. No format specification should be given that provides for more characters than permitted for a relevant input/output record. Thus, a format for a ASCII record to be printed out on the teletype should not provide for more characters than the capabilities of the printer on that model.

3. Information to be transmitted with O- and G- conversion may have real or integer names; information to be transmitted with E- and F- conversion must have real names; information to be transmitted with I- conversion must have integer names.

4. For G-conversion input values will be stored as floating point numbers. G-conversion is output in F- or E-conversion depending on the range.

5. The field width w, for E-, F-, and G-conversion, must include a space for the decimal point and a space for the sign. E- and G-conversion also require space for the exponent. Thus, for E- conversion, $w \geq d + 7$ and for F-conversion, $w \geq d + 3$.

6. The exponent, which may be used with E-conversion, is the power of 10 to which the number must be raised to obtain its true value. The exponent is written with an E (for E-conversion) followed by a minus sign if the exponent is negative, or a plus sign or a blank if the exponent is positive, and then followed by two numbers that are the exponent. For example, the number .002 is equivalent to the number .2E-02.

7. E- or F- conversion may be used for floating point numbers whose absolute value is less than 2^{27}. E-conversion must be used for numbers whose absolute value is greater than or equal to 2^{27} (134,217,728).

8. If a number converted by I-conversion requires more spaces than are allowed by the field width w, the excess on the high-order side is lost. If the number requires fewer than w spaces, the leftmost spaces are filled with blanks. If the number is negative, the space preceding the leftmost digit will contain a minus sign if sufficient spaces have been reserved.

9. If an output number that is converted by E-, F-, G-, or I-conversion requires more spaces than are allowed by the field width w, the most significant part of the number is truncated to fit the field. If the number requires fewer than w spaces, the leftmost spaces are filled with blanks. The output field is filled with blanks if the value of the output number is +377777777777 (octal).

Alphameric Fields

In addition to ASCII and FILENAME provided by FORTRAN, there are two ways by which alphameric information may be transmitted; both specifications result in storing the alphameric information internally in ASCII.

1. The specification Aw causes w characters to be read into, or written from, a variable or array name.

2. The specification nH introduces alphameric information into a FORMAT statement.

The basic difference between A- and H-conversion is that information handled by A-conversion is given a variable name or array name that can be referred to for processing and modification, whereas, information handled by H-conversion is not given a name and may not be referred to or manipulated in storage in any way.

A-Conversion

The variable name to be converted by A-conversion must conform to the normal rules for naming FORTRAN variables; it may be any type of variable.

1. On input, nAw will be interpreted to mean that the next n successive fields of w characters each are to be stored as ASCII information. If w is greater than four, the characters will be left-adjusted, and the word filled out with blanks.

2. On output, nAw will be interpreted to mean that the next n successive fields of w characters each are to be the result of transmission from storage without conversion. If w exceeds four, only four characters of output will be transmitted, preceded by w-4 blanks. If w is less than four, the w leftmost characters of the word will be transmitted.

3. If the variable involved is a filename, all above references to four become eight.

H-Conversion

The specification nH is followed in the FORMAT statement by n alphameric characters. For example:

```
31HbTHISbISbALPHAMERICbINFORMATION
```

Note that blanks are considered alphameric characters and must be included as part of the count n. The effect of nH depends on whether it is used with input or output.

1. On input, n characters are extracted from the input record and replace the n characters included with the source program FORMAT specification.

2. On output, the n characters following the specification, or the characters that replace them, are written as part of the output record.

Figure 3 is an example of A- and H-conversion in a FORMAT statement. The statement FORMAT (4HbXY=, F8.3,A4) might produce the following lines, where ƀ indicates a blank character:

```
XY=ƀ-93.210 mins
XY=9999.999 secs
XY=ƀƀ28.768 hrs
```

Figure 3

3. Quotation marks may be used in place of the nH specification for alphameric characters. The first example would then be written:

   "THIS IS ALPHAMERIC INFORMATION"

Logical Fields

Logical variables may be read or written by means of the specification Lw, where L represents the logical type of conversion and w is an integer constant that represents the data field width.

1. On input, a value of .TRUE. will be stored if the field of w characters is .TRUE. or if the first nonblank character in the field is T. A value of .FALSE. will be stored if the field of w characters is .FALSE., if the first nonblank character in the field is an F, or if all the characters are blank.

2. On output, a value of .TRUE. or .FALSE. in storage will cause w-1 blanks, followed by a T or an F, respectively, to be written.

Blank Fields -- X-Conversion

The specification nX introduces n blank characters into an input/output record.

1. On input, nX causes n characters in the input record to be skipped, regardless of what they are.

2. On output, nX causes n blanks to be introduced into the output record.

Repetition of Field Format

It may be desired to print or read n successive fields in the same format within one record. This may be specified by giving n, an unsigned integer, before E, F, G, I, L, O, or A. Thus, the field specification 3E12.4 is the same as writing E12.4, E12.4, E12.4.

Repetition of Groups

A limited parenthetical expression is permitted to enable repetition of data fields according to certain format specifications within a longer FORMAT statement. Thus, FORMAT (2(F10.6, E10.2),I4) is equivalent to FORMAT (F10.6, E10.2, F10.6, E10.2, I4). (See "Multiple-Record Formats" below.)

Scale Factors

To permit more general use of E-, F-, and G-conversion, a scale factor followed by the letter P may precede the specification. The magnitude of the scale factor must be between -8 and +8, inclusive. The scale factor is defined for input as follows:

$$10^{-\text{scale factor}} \times \text{external quantity} = \text{internal quantity}$$

The scale factor is defined for output as follows:

$$\text{external quantity} = \text{internal quantity} \times 10^{\text{scale factor}}$$

For input, scale factors have affect only on F-conversion. For example, if input data is in the form xx.xxxx and it is desired to use it internally in the form .xxxxxx, then the FORMAT specification to effect this change is 2PF7.4. For output, scale factors may be used with E-, F-, and G-conversion.

For example, the statement FORMAT (I2, 3F11.3) might give the following printed line:

```
27ƀƀƀƀ-93.209ƀƀƀƀƀ-0.008ƀƀƀƀƀƀ0.554
```

But the statement FORMAT (I2, 1P3F11.3) used with the same data would give the following line:

```
27ƀƀƀ-932.094ƀƀƀƀƀ-0.076ƀƀƀƀƀƀ5.536
```

Whereas, the statement FORMAT (I2, -1P3F11.3) would give the following line:

```
27ƀƀƀƀƀ-9.321ƀƀƀƀƀ-0.001ƀƀƀƀƀƀ0.055
```

A positive scale factor used for output with E-conversion increases the number and decreases the exponent. Thus, with the same data, FORMAT (I2, 1P3E12.4) would produce the following line:

```
27b̸-9.3209Eb̸01b̸-7.5804E-03b̸b̸5.5536E-01
```

The scale factor is assumed to be zero if no other value has been given. However, once a value has been given, it will hold for all E-, F-, an G-conversions following the scale factor within the same FORMAT statement. This applies to both single-record formats and multiple-record formats. Once the scale factor has been given, a subsequent scale factor of zero in the same FORMAT statement must be specified by 0P. For F-type conversion, output may not include numbers whose absolute value is greater than or equal to 2^{35} after scaling. Such numbers will be output in E-conversion. Scale factors have no affect on I- and O-conversion.

Multiple-Record Formats

To deal with a block of more than one line of print, a FORMAT specification may have several different one-line formats separated by a slash to indicate the beginning of a new blank line. Thus, FORMAT (3F9.2,2F10.4/8E14.5) would specify a multiline block of print in which lines 1, 3, 5, format (3F9.2,2F10.4), and lines 2, 4, 6, ... have format (8E14.5).

If multiple-line format is desired in which the first two lines are to be printed according to a special format and all remaining lines according to another format, the last line-specification should be enclosed in a second pair of parentheses;e.g.,FORMAT(I2,3E12.4/2F10.3,3F9.4/(10F12.4)). If data items remain to be transmitted after the format specification has been completely "used", the format repeats from the last previous parenthesis, which is a zero or a first-level parenthesis.

```
FORMAT (3E10.3, (I2,2(F12.4,F10.3)),E18.8)
       0        1 2              21     0
```

The parentheses labeled 0 are 0-level parentheses; those labeled 1 are first-level parenthesis; and, those labeled 2 are second-level parentheses. If more items in the list are to be transmitted after the format statement has been completely used, the FORMAT repeats from the last first-level left parenthesis; i.e., the parenthesis preceding I2.

As these examples show, both the slash and the final right parenthesis of the FORMAT statement indicate a termination of a record.

Blank lines may be introduced into a multiline FORMAT statement by listing consecutive slashes. When n+1 consecutive slashes appear at the end of the FORMAT, they are treated as follows: for input, n+1 records are skipped; for output, n blank lines are written. When n+1 consecutive slashes appear in the middle of the FORMAT, n records will be skipped for input and n blank lines are written for output.

FORMAT Statements Read at Object Time

FORTRAN accepts a variable FORMAT address. This permits specifying a FORMAT for an input/output list at object time.

```
     DIMENSION FMT (4)
1    FORMAT (4A4)
     READ (FILE1,1) (FMT(I), I=1,4)
     READ (FILE1,FMT) A, B, (C(J),J=1,5)
```

Figure 4

In Figure 4, A, B, and the array C are converted and stored according to the FORMAT specifications read into the array FMT at object time.

1. The name of the variable FORMAT specification must appear in a statement with dimension information, even if the array size is only 1.

2. The format read in at object time must take the same form as a source program FORMAT statement, except that the word FORMAT is omitted; i.e., the variable format begins with a left parenthesis.

Example: In Figure 4 above, the following format might be read in for FMT:

```
(2E14.6,5F6.2)
```

Then in the READ (FILE1, FMT) statement, A and B would have the format E14.6 and the five values for C would have the format F6.2.

Data Input Referring to a FORMAT Statement

Data input to the object program is typed or read from paper tape according to the following specifications:

1. The data must correspond in order, type, and field with the field specifications in the FORMAT statement.

2. Plus signs may be omitted or indicated by a +. Minus signs must be indicated.

3. A blank in a numeric field is treated as a zero. A numeric field containing all blanks is converted to zero.

4. Numbers for E- and F-conversion may contain any number of digits, but only the high-order eight digits of precision will be retained.

5. For input, numeric data must be situated at the extreme right of its field (right-justified).

Certain relaxations in input data format are permitted.

1. Numbers for E-conversion need not have four columns devoted to the exponent field. The start of the exponent field must be marked by an E or, if that is omitted, by a plus or minus sign (not a blank). Thus, E2, E+2, +2, and +02 are all permissible exponent fields.

2. Numbers for E-, and F-conversion need not have a decimal point; the format specification will supply it. For example, the number -09321+2 with the specification E12.4 will be treated as though the decimal point was between the zero and the nine. If the decimal point is in the line of data, its position overrides the position indicated in the FORMAT specification.

MEMORY-TO-MEMORY DATA CONVERSION STATEMENTS

Two statements which are associated with formatted READ and WRITE statements are DECODE and ENCODE, respectively. In an ENCODE/DECODE operation, no actual input/output takes place; data conversion and transmission takes place between an internal buffer area and the elements specified by a list. This buffer area is designated by the programmer and is usually an array. When multiple records are specified by the FORMAT being used, records after the first record follow each other in the buffer area.

ENCODE Statement

General Form

ENCODE (a,n) list

where:

1. n is either the statement number or the array name of the FORMAT statement describing the data being encoded.
2. a is an array name which specifies the starting location of the internal buffer.
3. List is as specified for a WRITE statement.

The ENCODE statement causes the data items specified by the list to be converted to character strings, according to the FORMAT specified by n, and placed in storage beginning at location a.

The number of characters for a record caused to be generated by the FORMAT statement and list should not be greater than the size of the array a.

Example:

The following example of ENCODE will print a series of squares made up of # signs. The number of squares printed can be varied by changing the value of the variable TIMES. By changing the contents of the list items, STAR and CENTER, different designs could be created. In this program, the squares are 16 # signs in width and 9 # signs in length. The variables J and K control the spacing between each printed line, and LEDGE controls the width of the left-hand margin. The data items in the list are stored in ARRAY, which is then printed line by line to form the squares. A sample square is shown.

```
      ASCII ARRAY(9),STAR(4),CENTER(4),J
      INTEGER TIMES
      STAR(1) = 4H####
      STAR(2) = 4H####
      STAR(3) = 4H####
      STAR(4) = 4H####
      CENTER(1) = 4H#ƀƀƀ
      CENTER(2) = 4Hƀƀƀƀ
      CENTER(3) = 4Hƀƀƀƀ
      CENTER(4) = 4Hƀƀƀ#
      LEDGE=5; LENGTH=7; J="-"; K=2
      TIMES=4
      DO 1Ø N=1, TIMES
      ENCODE (ARRAY,3Ø)J,LEDGE,(STAR(I),I=1,4)
      PRINT ARRAY
      DO 2Ø M=1,LENGTH
      ENCODE (ARRAY,3Ø)K,LEDGE,(CENTER(L),L=1,4)
      PRINT ARRAY
2Ø    CONTINUE
      ENCODE (ARRAY,3Ø)K,LEDGE,(STAR(I),I=1,4)
      PRINT ARRAY
1Ø    CONTINUE
3Ø    FORMAT (3H(1H,A1,1H,I2,5HX,16H,4A4,1H))
      STOP
      END
```

```
################
#              #
#              #
#              #
#              #
#              #
#              #
#              #
################
```

DECODE Statement

General Form

DECODE (a,n) list

where:

1. n is either the statement number or the array name of the FORMAT statement describing the data being decoded.
2. a is an array name which specifies the starting location of the internal buffer.
3. List is as specified for a READ statement.

The DECODE statement causes the character string beginning at location a to be converted to data items, according to the FORMAT specified by n, and stored in the elements of the list. The character string in the array consists of 9-bit ASCII characters.

The FORMAT statement and list should not require more characters than are in the array a. A new record is begun when specifically requested by the FORMAT.

Example:

Assume that a record has been read into contiguous character positions in array R:

```
 3        DECODE (R,5)J
 5        FORMAT (I1)
          GO TO(11,12,13,14,15,16,17,18,19),J
11        DECODE (R,21)(A(I), I=1,10)
21        FORMAT (1X, 10F9.3)
          GO TO 31
12        DECODE (R,22) K1,K2,K3,K4
22        FORMAT (1X,4I5)
          GO TO 32
13        DECODE (R,23) X,Y,Z
23        FORMAT (1X,3E20.9)
          etc.
              .
              .
              .
```

Explanation:

These statements illustrate a method of processing randomly ordered input records of varying format and data content. The type is identified by a digit from one to nine in the first column. Statement 3 converts the digit from character form to integer form. The GO TO then transfers to the DECODE/FORMAT combination prepared to process the specified format.

BINARY INPUT/OUTPUT STATEMENTS

The following table gives the forms of binary input/output statements. In the examples given, f is a file name, r is an integer expression or constant specifying the relative record in random file f, and n_2 is a statement number to which transfer is made upon reaching the end-of-file. No FORMAT statement is referenced and binary information frequently consists of large groups of numeric data.

General Form	Type of Input/Output
READ (f,END =n_2) list	Binary file input
READ (f'r) list	Random file input
WRITE (f) list	Binary file output
WRITE (f'r) list	Random file output

Examples:

```
READ (ABCDE,END = 50)(A(J),J = 1,10)
READ ("CHECK" ' 92) ARRAY
WRITE (FILE) JLB,KN, (B(L),L = 1,8)
WRITE (MATRIX' 15) DET, TRANS, INVT
```

The READ (f,END = n_2) list statement causes binary information to be read from file f. If the end-of-file is reached, a transfer is made to statement n_2.

The READ (f'r) list statement causes binary information to be read from record r of random file f.

The WRITE (f) list statement causes binary information to be written on file f.

The WRITE (f'r) list statement causes binary information to be written on record r of random file f. A random file is written as a fixed number of records with each record having a fixed number of 4-character words.

In three of the examples given above the names of the files, ABCDE, FILE, and MATRIX, are filename variables. They must be equated to filename constants enclosed in quotes before they appear in an input/output statement. The constants will then be the actual names of files used in the READ or WRITE statements. As an alternate method to this, the names of the files may be enclosed in quotation marks in the input/output statements. An example is given below where either method 1 or method 2 is valid.

1. WRITE ("MATRIX") list

2. FILENAME ARRAY
   ARRAY = "MATRIX"
   WRITE (ARRAY) list

NAMELIST INPUT/OUTPUT STATEMENTS

The four input/output statements which reference a NAMELIST name are given below.

General Form	Type of Input/Output
READ x	ASCII terminal input
READ (f,x,END = n_2)	ASCII file input
PRINT x	ASCII terminal output
WRITE (f,x)	ASCII file output

In the given forms f is a file name, x is the NAMELIST name, and n_2 is a statement number to which transfer is made upon reaching the end-of-file.

Examples:

```
          READ ("FILE1", NAM1,END = 45)
          WRITE ("FILE2",NAM2)
```

The READ (f,x,END = n_2) statement causes ASCII information related to variables and arrays associated with the NAMELIST name x to be read from file f. Transfer is made to statement n_2 when the end-of-file is reached. END = n_2 may be omitted from the statement.

The WRITE (f,x) statement causes ASCII information related to variables and arrays associated with the NAMELIST name x to be written on file f.

NAMELIST Statement

The NAMELIST statement and the above forms of the READ and WRITE statements provide for reading, writing, and converting data without the use of an input/output list in the input/output statement and without a reference to a FORMAT statement.

General Form

NAMELIST /X/A,B...C/Y/D,E,....F/Z/G,H,...I

where:

1. X,Y,Z,...are NAMELIST names.
2. A,B,C,D,...are variable or array names.

Examples:

```
     DIMENSION A(10), I(5,5), L(10)
     NAMELIST /NAM1/A,B,I,J,L/NAM2/A,C,J,K
```

In the preceding examples, the arrays A, I, and L and the variables B and J belong to the NAMELIST name NAM1, and the array A and the variables C, J, and K belong to the NAMELIST name, NAM2.

Each list that is mentioned in the NAMELIST statement is given a NAMELIST name. Only the NAMELIST name is needed in an input/output statement to refer to that list thereafter in the program. The following rules apply to assigning and using a NAMELIST name:

1. A NAMELIST name consists of one to eight alphameric characters; the first character must be alphabetic.

2. A NAMELIST name is enclosed in slashes when defined in a NAMELIST statement.

The field of entries belonging to a NAMELIST name ends either with a new NAMELIST name enclosed in slashes or with the end of the NAMELIST statement.

3. A variable name or any array name may belong to one or more NAMELIST names.

4. A NAMELIST name must not be the same as any other name in the program.

5. A NAMELIST name may be defined only once by its appearance in a NAMELIST statement. After it has been defined in the NAMELIST statement, the NAMELIST name may appear only in READ or WRITE statements thereafter in the program.

6. A NAMELIST statement defining a NAMELIST name must precede any appearance of the name in the program, and must follow all the specification statements such as COMMON, DIMENSION, etc.

7. A dummy argument which appears in a FUNCTION or SUBROUTINE statement cannot be used as a variable in a NAMELIST statement.

8. If a NAMELIST statement contains a dimensioned variable, the statement that contains the dimension information defining the variable must precede the NAMELIST statement.

Data Input Referring to a NAMELIST Statement

When a READ statement refers to a NAMELIST name, the designated input device is prepared and input of data is begun. The first input data record is searched for a $ as the first nonblank character, immediately followed by the NAMELIST name, immediately followed by a comma or one or more blank characters. When a successful match is made of the NAMELIST name on a data record and the NAMELIST name referred to in a READ statement, data items are converted and placed in storage.

Any combination of four types of data items, described in the following text, may be used in a data record. The data items must be separated by commas. If more than one record is needed for input data, the last item of each record, except the last record, must be a constant followed by a line feed or carriage return. The end of a group of data is signaled by $ either in the same data record as the NAMELIST name or anywhere in any succeeding records.

The form that data items may take is:

1. Variable name = constant

   where variable name may be an array element name or a simple variable name.

2. Array name = set of constants (separated by commas)

   where k* constant may be included to represent k constants (k must be an unsigned integer).

3. Subscripted variable = set of constants (separated by commas)

   where k* constant may be included to represent k constants (k must be an unsigned integer). A data item of this form results in the set of constants being placed in consecutive array elements, starting with the element designated by the subscripted variable. The number of constants given cannot exceed the number of elements in the array that are included between the given element and the last element in the array, inclusive.

4. Variable 1/Variable 2 = constant

   where Variable 1 is a counter which is set after the data has been input, indicating the number of constants that have been stored for Variable 2. Variable 1 is a nondimensioned integer name and Variable 2 is a dimensioned array name.

Constants used in the data items may take any of the following forms:

a. integers
b. real numbers
c. logical constants, which may be written as .TRUE. and .FALSE. or T and F
d. ASCII constants
e. filename constants

Logical constants may be associated only with logical variables. The other types of constants may be associated with the other variables and are converted in accordance with the type of variable. Blanks must not be embedded in a constant or repeat constant field, but may be used freely elsewhere within a data record.

Any selected set of variable or array names belonging to the NAMELIST name that is referred to by the READ statement may be used as specified in the preceding description of data items.

Examples:

```
First Data Record     $NAM1 I(2,3)=5,J=4.2, B=4
Second Data Record    A(3)=7, 6.4, L=2, 3, 8*4.3$

                                 or

                      $NAM1 I(2,3)=5, J=4.2, B=4
                     =A(3)=7, 6.4, L=2, 3, 8*4.3$
```

If this data is input to be used with the NAMELIST statement previously illustrated and with a READ statement, the following actions take place. The input unit designated in the READ statement is prepared and the first record is read. The record is searched for a $, immediately followed by the NAMELIST name, NAM1. Since the search is successful, data items are converted and placed in core storage.

The integer constant 5 is placed in I(2,3), the real constant 4.2 is converted to an integer and placed in J, and the integer constant 4 is converted to real and placed in B. Since no data items remain in the record, the next input record is read. The integer constant 7 is converted to real and placed in A(3), and the real constant 6.4 is placed in the next consecutive location of the array, A(4). Since L is an array name not followed by a subscript, the entire array is filled with the succeeding constants. Therefore, the integer constants 2 and 3 are placed in L(1) and L(2), respectively, and the real constant 4.3 is converted to an integer and placed in L(3), L(4),..., L(10). The $ signals termination of the input for the READ operation.

If an array is not filled by the end of a line and the line ends with a comma, a zero will be placed in the next location of the array. To avoid this, the last comma should be omitted. For example, if C is dimensioned 5, the following line would make C(5) = Ø:

```
          C(1)=2.3,4.1,1.7,6.2,
```

To avoid this, the line should be as follows:

```
          C(1)=2.3,4.1,1.7,6.2
```

MANIPULATIVE INPUT/OUTPUT STATEMENTS

File Manipulation

General Form

```
BEGIN FILE f
END FILE f
CLOSE FILE f
BACKSPACE f
OPEN FILE f (password)
```

Examples:

```
BEGIN FILE test         BACKSPACE quad
END FILE product        CLOSE FILE joe
OPEN FILE sam ("george")
```

The BEGIN FILE f statement resets file f to the beginning of the first record.

The END FILE f statement terminates file f with a logical end-of-file. This statement can be used to truncate and then append to, rather than replace records on, an existing file.

The BACKSPACE f statement backspaces file f one logical record.

The CLOSE FILE f statement closes file f and releases its buffer.

END FILE, BEGIN FILE, and BACKSPACE may not be used with random files.

The OPEN FILE f (password) statement permits a user automatically to supply the password for a passworded file. The password given in parentheses is a filename variable or constant (if the latter, enclose in quotes).

Referencing a File

To reference a file for the first time, or to use a file that was previously created, the user should list the name of the file at the beginning of his program in a Filename-Type statement. Once he has done this, the user may write directly on the file, or may read it, or truncate it with an END FILE statement and then add more records. In place of a Filename-Type statement, it is also possible to reference a file using a filename constant in quotation marks. The following two examples illustrate these alternate methods:

Filename Constant	Filename Variable

`WRITE ("JOE", 1Ø)A,B,C*D`	`FILENAME JOE`
	`JOE = "BILL"`
	`WRITE (JOE,1Ø)A,B,C*D`

In the first case, JOE is a filename constant because it is enclosed in quotes and used in an input/output statement.

Note

If the filename constant is used in quotes and in a CALL statement, it must be at least five characters long.

Example: `CALL RSUBR ("SECOND",A,K)`

In the second case, JOE is typed as a filename variable. Then it is replaced by the filename constant BILL, so that at the time of the WRITE statement the file used will be BILL rather than JOE.

In both cases the file names will be stored as two words:

JOEb bbbb or BILL bbbb.

END-OF-FILE Test and Branch

Testing and branching on end-of-file is accomplished by the following READ statements:

for binary records	READ (f, END = n_2) list
for ASCII records	READ (f,n,END = n_2) list

In each case, n_2 is the statement number to which transfer is made upon reaching the end-of-file.

The end-of-file is logical rather than being a physical mark or space. Hence, the following sequences of statements are permitted:

```
END FILE JOE
BACKSPACE JOE
READ (JOE) etc.
```

These instructions will terminate and then read the preceding record on the file.

Also

```
END FILE JOE
WRITE (JOE) etc.
```

which will truncate (e.g., an input file) and then append a record to the file.

MODIFYING AN EXISTING FILE

The input/output routines permit the user to replace individual records of an existing file. All random file WRITE statements are considered replacements. A linked file WRITE statement which will modify one or more existing records is also considered replacement and the record or records being written must be the same length as the ones which existed previously. If not, execution will be terminated, the previously existing records may or may not have been modified, and the file can be subsequently processed with a valid WRITE statement. The END FILE statement can be used when it is desired to truncate an existing file and add rather than replace information.

FILE CONVENTIONS

1. ASCII records do not have a fixed record length (e.g., 72, 80, 120, etc.). The record length is defined by the FORMAT statement when used, and if not, by the I/O list.

2. A record on an existing file may be replaced by a WRITE of a record of exactly the same size. If the record being written is not the same size, the user will be given a message and aborted.

3. A user may have from one to five program-I/O files open at any one time.

4. Data files created by an object program are not automatically saved in the permanent file system. (Either the PERM or BYE command may be used to achieve this.)

9. Subroutine, Function, and Subprogram Statements

There are four classes of subroutines in FORTRAN: arithmetic statement functions, built-in functions, FUNCTION subprograms, and SUBROUTINE subprograms. The major differences among the four classes of subroutines are as follows:

1. The first three classes may be grouped as functions; they differ from the SUBROUTINE subprograms in the following respects:

    a. The functions are always single-valued (i.e., they return only a single result); the SUBROUTINE subprogram may return more than one value.

    b. A function is referred to by an arithmetic expression containing its name; a SUBROUTINE subprogram is referred to by a CALL statement.

2. The built-in function is an open subroutine; i.e., a subroutine that is incorporated into the object program each time it is referred to in the source program. The three other FORTRAN subroutines are closed; i.e., they appear only once in the object program.

NAMING SUBROUTINES

In the following text, the terms "calling program" and "called program" are used. The calling program is the program in which a subroutine is referred to or called. The called program is the subroutine that is referred to or called by the calling program.

All four classes of subroutines are named in the same manner as a FORTRAN variable (see "Variables", in Chapter 4).

1. A subroutine name may be any length, but only the first eight characters are used for identification.

2. The type of function, which determines the type of the result, may be defined as follows:

   a. The type of an arithmetic statement function may be indicated by the name (if it is real or integer) of the function or by placing the name in a Type statement.

   b. A FUNCTION subprogram is defined by placing its name after the word FUNCTION in a Type statement. The type of the FUNCTION is indicated by putting the name in a Type statement immediately following the FUNCTION definition. For example:

      ```
      FUNCTION LOAN (C)
      REAL LOAN
      ```

      The type of a reference to a FUNCTION subprogram in the Subroutine Library (the mathematics subroutines) is automatically defined as shown in Figure 6. Therefore, the subprogram need not be typed in the calling program.

   c. The type of a built-in function is indicated within the FORTRAN processor and need not appear in a Type statement (see column 6 of Figure 5).

3. The name of a subroutine subprogram has no type and should not be defined, since the type of results returned is dependent only on the type of the variable names in the dummy argument list.

DEFINING SUBROUTINES

The method of defining each class of subroutines is discussed below.

Arithmetic Statement Functions

Arithmetic statement functions are defined by a single arithmetic statement and apply only to the source program containing the definition.

<u>General Form</u>

a = b

where:

1. a is a function name followed by parentheses enclosing its arguments, which must be distinct, nonsubscripted variables, separated by commas.
2. b is an expression that may involve subscripted variables. Any arithmetic statement function appearing in b must have been previously defined.

Examples:

```
FIRST (X) = A*X+B

JOB (X,B) = C*X+B

THIRD (D) = FIRST (E)/D

LOGFCT (A,C) = A**2.GE.C/D

MAX (A,I) = A**I-B-C
```

1. As many as desired of the variables appearing in b may be stated in a as the arguments of the function. Since the arguments are dummy variables, their names, which indicate the type of the variable, may be the same as names appearing elsewhere in the program of the same type.

2. Those variables included in b that are not stated as arguments to this arithmetic statement function are the parameters of the function. They are ordinary variables.

3. All arithmetic statement function definitions must precede the first executable statement of the source program.

   The only statements which may precede an arithmetic statement function are the following:

   FUNCTION
   SUBROUTINE
   COMMON
   DIMENSION
   INTEGER
   REAL
   LOGICAL
   ASCII
   FILENAME
   EXTERNAL

4. The type of any arithmetic statement function name or argument that differs from its implicit type must be defined preceding its use in the arithmetic statement function definition.

Built-In Functions

Built-in functions are pre-defined subroutines that exist within the FORTRAN processor. A list of all the available built-in functions is given in Figure 5. An example of the use of each of the built-in functions is given in Appendix E.

Function	Definition	No. of Args.	Name	Type of Argument	Type of Function
Absolute	$\|Arg\|$	1	ABS IABS	Real Integer	Real Integer
Trun-cation	Sign of Arg times largest integer $\leq \|Arg\|$	1	AINT INT	Real Real	Real Integer
Remain-dering	Arg_1 (mod Arg_2)	2	AMOD MOD	Real Integer	Real Integer
Choosing largest value	Max (Arg_1, Arg_2,...)	≥ 2	AMAX0 AMAX1 MAX0 MAX1	Integer Real Integer Real	Real Real Integer Integer
Choosing smallest value	Min (Arg_1), Arg_2,...)	≥ 2	AMIN0 AMIN1 MIN0 MIN1	Integer Real Integer Real	Real Real Integer Integer
Float	Coversion from integer to real	1	FLOAT	Integer	Real
Fix	Coversion from real to integer with truncation	1	IFIX	Real	Integer
Transfer Sign	Sign of Arg_2 times $\|Arg_1\|$	2	SIGN ISIGN	Real Integer	Real Integer
Positive differ-	Arg_1 - Min (Arg_1,Arg_2)	2	DIM IDIM	Real Integer	Real Integer

Figure 5

Note

The function MOD(Arg_1,Arg_2) is defined as Arg_1 -(Arg_1/Arg_2)Arg_2 where (Arg_1/Arg_2) is the truncated value of that quotient.

FUNCTION Subprograms

FUNCTION subprograms are defined by a special FORTRAN source language program.

<u>General Form</u>

FUNCTION name (a_1,a_2,...,a_n)

REAL name

where:

1. Name is the symbolic name of a single-valued function.
2. The arguments a_1,a_2,...,a_n, of which there must be at least one, are nonsubscripted variable names or the dummy name of a SUBROUTINE or FUNCTION subprogram.
3. The type of the function may be explicitly stated following the FUNCTION definition, such as REAL above.

Examples:

```
FUNCTION ARCSIN (RADIAN)

FUNCTION ROOT (A,B,C)

FUNCTION CONST (NG,JG)
INTEGER CONST

FUNCTION IFTRU (D,E,F)
LOGICAL IFTRU
```

1. The FUNCTION statement must be the first statement of a FUNCTION subprogram.

2. The name of the function must appear at least once as a variable on the left side of an arithmetic statement or in an input statement. This name cannot be used in a NAMELIST statement.

3. The last value of the function is one which is returned.

   For example:

   ```
   FUNCTION CALC (A,B)
   .
   .
   .
   CALC = Z + B
   .
   .
   .
   RETURN
   .
   .
   .
   END
   ```

   By this means, the output value of the function is returned to the calling program.

4. The arguments may be considered dummy variable names that are replaced at the time of execution by the actual arguments supplied in the function reference in the calling program. The actual arguments must correspond in number, order, and type with the dummy arguments.

5. When a dummy argument is an array name, a statement with dimension information must appear in the FUNCTION subprogram; the corresponding actual argument must be a dimensioned array name.

6. The FUNCTION subprogram must be logically terminated by a RETURN statement (see "Normal Returns from Subprograms", below).

7. The FUNCTION subprogram may contain any FORTRAN statements except SUBROUTINE or another FUNCTION statement.

8. The actual arguments of a FUNCTION subprogram may be any of the following:

   a. Any type of constant.

   b. Any type of subscripted or nonsubscripted variable.

   c. An arithmetic or logical expression.

   d. The name of a FUNCTION or SUBROUTINE subprogram.

9. A FUNCTION subprogram is referenced by using its name as an operand in an arithmetic expression.

   Those FUNCTION subprograms that are supplied with FORTRAN are given in Figure 6.

SUBROUTINE Subprograms

SUBROUTINE subprograms are defined by a special FORTRAN source language program.

General Form

SUBROUTINE name ($a_1,a_2,\ldots,a_n$) or SUBROUTINE name

where:

1. Name is the symbolic name of a subprogram.

2. Each argument, a, if any, is a nonsubscripted variable name or the dummy name of a SUBROUTINE or FUNCTION subprogram.

Examples:

```
    SUBROUTINE MATMPY (A, N, M, B, L, J)

    SUBROUTINE QUADEQ (B, A, C, ROOT1, ROOT2)

    SUBROUTINE OUTPUT
```

1. The SUBROUTINE statement must be the first statement of a SUBROUTINE subprogram.

2. The SUBROUTINE subprogram may use one or more of its arguments to return output. The arguments so used must appear on the left side of an arithmetic statement or in an input list within the subprogram.

3. The arguments may be considered dummy variable names that are replaced at the time of execution by the actual arguments supplied in the CALL statement, which refers to the SUBROUTINE subprogram. The actual arguments must correspond in number, order, and type with the dummy arguments.

4. When a dummy argument is an array name, a statement containing dimension information must appear in the SUBROUTINE subprogram; the corresponding actual argument in the CALL statement must be a dimensioned array name.

5. No argument in a SUBROUTINE statement may also be included in COMMON.

6. The SUBROUTINE subprogram must be logically terminated by a RETURN statement.

7. The SUBROUTINE subprogram may contain any FORTRAN statements except FUNCTION or another SUBROUTINE statement.

Normal Returns from Subprograms

The normal exit from any subprogram is the RETURN statement, which returns control to the calling program. The RETURN statement is the logical end of the program; there may be any number of RETURN statements in the program.

<u>General Form</u>

RETURN

Nonstandard Returns From SUBROUTINE Subprograms

The normal sequence of execution following the RETURN statement of a SUBROUTINE subprogram is to the next executable statement following the CALL statement in the calling program. It is also possible to return to any numbered executable statement in the calling program by using a special return from the called subprogram. This return may not violate the transfer rules for DO loops.

The following text describes the form of the FORTRAN statements that is required to return from the subroutine to a statement other than the next executable statement following the CALL.

The general form of the CALL statement in the calling program is:

<u>General Form</u>

CALL subr (a_1,a_2,...,a_n)

where:

1. Subr is the name of the SUBROUTINE subprogram being called.
2. a_i is a dummy argument of the form described under "CALL Statement", or is of the form:

   $n

   where n is a statement number, $ is the character $.

The general form of the SUBROUTINE statement in the called program is:

<u>General Form</u>

SUBROUTINE subr (a_1,a_2,...,a_n)

where:

1. Subr is the name of the subprogram.
2. a_i is a dummy argument of the form described under "SUBROUTINE Subprogram", or is of the form:

   *

   where * is the character asterisk (*) and denotes a nonstandard return.

The general form of the RETURN statement in the called program is:

General Form

RETURN i

where:

i is an integer constant or variable which denotes the ith nonstandard return in the argument list, reading from left to right.

Example:

Calling Program	Called Program
```
        .
        .
        .
10 CALL SUB (A,B,C,$30,$40)
20 ---
        .
        .
        .
30 ---
        .
        .
        .
40 ---
        .
        .
        .
      END
```

```
Calling Program                    Called Program

        .                          SUBROUTINE SUB (X,Y,Z,*,*)
        .                                .
        .                                .
10 CALL SUB (A,B,C,$30,$40)              .
20 ---                             100  IF (R) 200,300,400
        .                          200  RETURN
        .                          300  RETURN1
        .                          400  RETURN2
30 ---                                  END
        .
        .
        .
40 ---
        .
        .
        .
      END
```

In the preceding example, execution of statement 10 in the calling program causes entry into subprogram SUB. If statement 100 is executed, the return to the calling program will be to statement 20, 30, or 40, if R is less than, equal to, or greater than zero, respectively.

Nonstandard returns may be best understood by considering that a CALL statement that uses the nonstandard return is equivalent to a CALL and a computed GO TO statement in sequence.

For example:

```
CALL NAME (P,$20,Q,$35,R,$22)
```

is equivalent to

```
CALL NAME (P,Q,R,I)
GO TO (20,35,22),I
```

where I is set to the value of the integer in the RETURN statement executed in the called subprogram. If the RETURN is blank or zero, a normal (rather than nonstandard) return is made to the statement immediately following the GO TO.

Similarly, the arguments in the associated SUBROUTINE statement correspond to the arguments in the CALL statement as follows:

```
SUBROUTINE NAME (S,*,T,*,U,*)
```

Subprogram Names As Arguments

FUNCTION and SUBROUTINE subprogram names may be the actual arguments of subprograms. To distinguish these subprograms names from ordinary variables when they appear in an argument list, they must appear in an EXTERNAL statement.

```
EXTERNAL SIN
CALL SUBR (Z,SIN,B)
```

CALL STATEMENT

The CALL statement is used to refer to a SUBROUTINE subprogram.

<u>General Form</u>

CALL subr $(a_1,a_2,\ldots,a_n)$

where:

1. Subr is the name of a SUBROUTINE subprogram.

2. $a_1,a_2,\ldots,a_n$ are the n arguments.

Examples:

```
     CALL MATMPY (X,5,10,Y,7,2)
     CALL QDRTIC (9.732,Q/4.536,R-S**2.0,X1,X2)
     CALL OUTPUT
```

The CALL statement transfers control to the subprogram and presents it with the actual arguments.

The arguments may be any of the following:

1. Any type of constant.

2. Any type of subscripted or nonsubscripted variable.

3. An arithmetic or logical expression.

4. Alphameric characters. Such arguments must be preceded by nH ($n \leq 8$) where n is the count of characters included in the argument; e.g., 4HLIST. Note that blank spaces and special characters are considered in the character count when used in alphameric fields. Alphameric characters may also be enclosed in quotation marks.

5. The name of a FUNCTION or SUBROUTINE subprogram.

The arguments presented by the CALL statement must agree in number, order, type, and array size (except as explained under the DIMENSION statement) with the corresponding arguments in the SUBROUTINE statement of the called subprogram.

MATHEMATICAL SUBROUTINES

Time-Sharing FORTRAN provides various commonly used mathematical subroutines, defined as FUNCTION subprograms. The names of all these subprograms are automatically typed by the FORTRAN compiler; therefore, they need not appear in Type statements.

Variables used as arguments of mathematical subroutines must be typed, either explicitly or implicitly, in accordance with the function in which they appear.

A list of the mathematical subroutines provided by FORTRAN is given in Figure 6. The range of argument required for each of these subroutines is given in Appendix F.

Function	Definition	Number of Args.	Name
Exponential	e^{Arg}	1	EXP
Natural logarithm	$\log_e$ (Arg)	1	ALOG
Trigonometric	sin (Arg)	1	SIN
Trigonometric cosine	cos (Arg)	1	COS
Trigonometric tangent	tan (Arg)	1	TAN
Trigonometric cotangent	cot (Arg)	1	COTAN
Arctangent	arctan (Arg) arctan(Arg_1/Arg_2)	1 2	ATAN ATAN2
Arcsine	arcsine(Arg)	1	ARSIN
Arccosine	arccosine(Arg)	1	ARCOS
Hyperbolic tangent	tanh(Arg)	1	TANH
Square root	$(Arg)^{\frac{1}{2}}$	1	SQRT

Arguments and functions for all subroutines are real.

Figure 6

Appendix A. Time-Sharing System Command Language

INTRODUCTION

The Time-Sharing System user has a choice of systems with which to solve his problems. The available systems are:

- BASIC - an algebraic-language compiler/executor designed for the casual user.
- FORTRAN - an algebraic-language compiler/loader with extended capabilities for subprogramming, chain overlays, peripheral I/O, etc., providing full batch-type programming capabilities.
- Text EDITOR - a system for building and maintaining text files of any description.
- CARDIN - a system for building batch-job files and submitting them to the GECOS-III processor.

These systems are controlled by means of a command language--a set of orders or instructions with which a user requests functions to be performed (e.g., LIST, RUN) and manages the flow of control for his session at the terminal (e.g., BYE, DONE).

The command language is common to the four systems of the time-sharing system, but with some variations in the applicability of a particular command to one or more of the systems. These variations are listed in the table "Applicability of Commands By System." Each command activates an associated subsystem which performs the function requested. The command subsystem activated will be the same regardless of system selection with but one exception--the RUN subsystem. But the initial selection of a system sets proper indicators to trigger the required path of control in the command subsystem and this exception is automatically compensated for.

Commands can only be given while a system is in "build-mode," a mode in which the system is expecting input. Build-mode is indicated by a system-supplied asterisk at the beginning of each new input line.

DEFINITIONS

- Line Numbers

  Line numbers are required by some systems for line sequencing purposes. In the case of BASIC, line numbers are also used as statement reference numbers. A line number consists of one to eight numeric characters, terminated by a nonnumeric, and preceded on the line only by blanks (if any exist).

- Manual Mode

  In manual mode, the user must provide (type) the line numbers for each line.

- Automatic Mode

  In automatic mode, the system provides the line numbers. They are printed as the build-mode request for input (asterisk) is issued. The number is written onto the collector file as a part of the statement.

- New File

  A new file is a temporary file created for the user when he responds NEW to any file request. It is assumed the user will build a file which then may be saved, thus creating an old file.

- Old File

  An old file is a previously built and saved file which the user selects by responding OLD to the file request and naming the file. The old file is copied onto the current file where it is available to the user for processing or modification.

- **Current File**

  The current file is a temporary file assigned to the user, on which a new file is built or on which the selected old file is copied. Regardless of the intervening commands or system selections, the current file contains the last NEW or OLD selection, with whatever modification may have been entered.

- **Collector File**

  The collector file is a temporary file assigned to each user when he logs on. All input which is not a recognizable command is gathered onto this file--for example, numbered statements. Then, when the file becomes full or a command is typed, depending upon the system, the collector file is merged with the current file and the entire current file sorted by line number. For example, when the commands RUN, LIST or SAVE are encountered in the BASIC system, if data exists in the collector file, it is merged with the current file in sort order. Note that the original old file, if any, will not be altered until a SAVE command naming that old file is executed.

- **Available File Table**

  An available file table (AFT) is provided for each time-sharing system user. This table holds a finite number of file names which are entered in the AFT when the files are accessed (opened). The advantages of the AFT are:

  1. Files requiring passwords or long catalog/file descriptions may be referenced by file name alone once they have been entered in the table.

  2. Files used repeatedly remain readily available, thus reducing the overhead time and cost of accessing the file each time.

The following commands cause the named permanent files to be placed in the AFT, if possible, and if they are not already there.

RUN	filename(s)
LIST	filename(s)
OLD	filename(s)
SAVE	filename(s)
GET	filename(s)
PRINT	filename(s)
PERM	tempfile, filename

Because the AFT is of a finite length, it can become full. When this happens and a command is given which requires a new filename to be placed in the AFT, the command subsystem will print an error message indicating that the AFT is full. At this point, the user must remove any unneeded files from the AFT in order to continue. The STATUS command produces a listing of all of the user's files in the AFT. The REMOVE command can be used to remove specified files from the AFT. The files are not purged or altered in any way; only the name is removed from the AFT and the file is set not-busy.

FILE FORMATS

The designation of files in the following discussion of commands will be in the following formats:

a. filename — where the file name only is required.

b. filedescr — where the full file description may be used, in any of the following formats:

1. filename

2. filename$password

3. userid/catalog$password...
   /catalog$password/filename$password

If a required password is not given (1), the system will explicitly ask for the password.

If a required password is omitted in the string format (3), a REQUEST DENIED message will be issued.

For cases (1) and (2), above: If the file was previously opened (e.g., with a GET), only the filename (1) need be given. If the requested file is not already open, it must emanate directly from the user's master catalog (quick-access type file).

Where desired-permissions and/or alternate-name are applicable, they are specified in the following format:

```
filedescr"altname",permissions

      (or, alternatively)

filedescr,permissions
```

where:

permissions may be any one or combination of the following, separated by commas:

READ (or R)

WRITE (or W)

EXECUTE (or E)

APEND (or A)

altname may be a valid file name (one to eight characters), enclosed in double-quote signs.

Note that where a desired-permissions specification is applicable, a null permissions field implies READ and WRITE permissions; i.e., the default interpretation for desired permissions is R,W.

If a file-segment specification, of the form (i,j) where i and j are line numbers, is given in addition to desired-permissions and/or alternate-name, it must appear last in the specification string; e.g.:

```
filedescr"altname", permissions(i,j)

                or

filedescr, permissions(i,j)
```

Examples:

```
OLD     FIL1$GOGO,R

SAVE    /CAT1CAT2$MAYI/FIL0$HERE

LIST    FILE2$HOHO

PURGE   FIL3$ARIZ;FIL4;FIL5$SUN

GET     JJONES/DATACAT/BATCHWRLDFIL"INFILE"
```

FILE NAMES, CATALOG NAMES, AND PASSWORDS

File names for time-sharing usage must be eight characters or less in length, and may be composed of alphanumerics, periods, and minus-signs. Catalog names and passwords may be up to 12 characters in length, and composed of the same characters as file names.

If a batch-world file with a name longer than eight characters (maximum 12 characters) is to be accessed, it must be given an alternate name ("altname") from one to eight characters in length.

COMMANDS

Following is a description of the time-sharing system commands. Although the command words are spelled out completely in the following descriptions, in actual usage those exceeding four characters may be shortened to four characters (e.g., RESEquence).

- NEW

A new file (empty current file) will be started. (The system will be in manual mode.) The current file is cleared of any prior contents.

- TAPE

The current file will be built or extended with input from paper tape. Neither line feeds nor line numbers are supplied by the time- sharing system. (The command is #TAP when in the EDITOR system.)

- OLD

1) OLD filedescr (permissions and alt-name applicable) File filedescr becomes the current file.

2) OLD filedescr(i,j) (permissions and alt-name applicable) Lines i through j of file filesdescr become the current file. Filedescr must be a line-numbered file.

3) OLD $\underline{f}(\underline{i},\underline{j})_1;\underline{f}(\underline{i},\underline{j})_2;\ldots\underline{f}(\underline{i},\underline{j})_n$ (permissions and alt-name applicable) The n files or file segments are adjoined in the order listed and become the current file, where f is a filedescr. Adjoining of BASIC files should be done with caution (sequence numbers are also statement numbers).

   Note that these file or segments are concatenated on the current file and resequencing may be required for satisfactory operation in line-number dependent systems. Sorting or resequencing is not automatic.

   If the file list is too long for one line, the OLD subsystem will request more input if a delimiter is the last non-blank character before the carriage return.

- LIB filename

File filename from the library becomes the current file.

- SAVE $\underline{\text{filedescr}}_1$;$\underline{\text{filedescr}}_2$;...;$\underline{\text{filedescr}}_n$

  The current file is saved on the permanent file(s) defined by filedescr. Sorting by line number is done or not done, according to system requirements. An alternate name ("altname") parameter may not be specified, but the altnamespecified for a permanent file previously opened, (e.g., an OLD or GET) must be used as the filename in the SAVE command. That is to say, if a file to be referred to in a SAVE command requires an alternate name for the reference, it must have been previously opened with "altname" specified. If the file does not already exist, it will be created if possible and general read permission assigned.

- PURGE $\underline{\text{filedescr}}_1$;$\underline{\text{filedescr}}_2$;...;$\underline{\text{filedescr}}_n$

  Delete the specified file(s) from the file system.

- REMOVE $\underline{\text{filename}}_1$;$\underline{\text{filename}}_2$;...;$\underline{\text{filename}}_n$

  Remove the specified file(s) from the AFT.

- PERM tempfile;filedescr

  The temporary file tempfile is copied onto the permanent file described by filedescr. If the file does not already exist, it will be created with general read permission. The temporary file name is removed from the AFT and the permanent file accessed (name placed in AFT).

- GET $\underline{\text{filedescr}}_1$;$\underline{\text{filedescr}}_2$;...;$\underline{\text{filedescr}}_n$
  (permissions and alt-name applicable)

  The permanent file(s) designated by filedescr will be accessed and the filename(s), or alternate names, if specified, placed in the AFT. This is a simple means by which common data files emanating from other user's master catalogs may be opened (using FORTRAN).

- RUN

1) RUN

   Execute the selected system. The source input is the current file. (If BASIC is the system selection and any variation of the RUN command is given, only the current file will be executed;i.e., any information appended after the RUN command is ignored.)

2) RUN filedescr (permissions and alt-name applicable)

   Under FORTRAN, compile and execute the file specified by filedescr.Under CARDIN, convert and pass the specified file to GEIN.

3) RUN = filedescr(opt$_1$,...,opt$_n$) (permissions and alt-name applicable if file already exists.)

   Under FORTRAN, compile and execute the current file using the specified options. Save the object program on the file specified by filedescr. If this file does not already exist, it will be created (with general- read permission) but only if it was specified as a quick-access file;i.e., emanating directly from the user's master catalog. Maximum size will be one link.

4) RUN filedescr$_1$;filedescr(i,j)$_2$;...;filedescr(i,j)$_n$ =filedescr$_x$ (opt,...,opt) (permissions and altname applicable to already existent files)

   Under FORTRAN, the specified files or file-segments are adjoined and compiled/executed according to the options specified, and the object program saved as file filedescr . The compile options and saving of object file are optional. The designated files may be object or source files. (Object files must be random files.)

   The current file may be indicated by an asterisk in the file list. Caution must be exercised to ascertain that the current file contains that which is expected.

If a list is too long to be typed on one line, the subsystem will request more input if a delimiter is the last nonblank character before the carriage return.

- CATALOG

1) CATAlog

   List all catalog and file names which emanate from the user's own master catalog.

2) CATAlog #LIB

   List all file names in the library.

3) CATAlog <u>filename</u>

   Print a list of the attributes of the file specified. The file must emanate from the user's catalog.

4) CATAlog /catalog1/catalog2

   Print a list of all catalog and file names which emanate from the specified catalog (catalog2 in this case).

5) CATAlog /catalog1/catalog2*

   Print a detailed list of catalog2's attributes.

Passwords need not be given in these catalog commands. However, CATALOG applies only to strings which originate from the user's (own) master catalog or the library (#LIB).

CATALOG may also be selected at system level.

- PRINT

Under CARDIN, print at the terminal all or any part of a source file, reformatting the file by use of format-option and/or tab characters, if desired.

1) PRINT

The entire current file will be printed.

2) PRINT filedescr($\underline{i},\underline{j}$)$_1$;filedescr($\underline{i},\underline{j}$)$_2$;...;filedescr($\underline{i},\underline{j}$)$_n$

The specified files or file-segments defined by filedescr will be adjoined, converted, and printed. The current file may be included in the string of files by the name *. The current file, however, will not be affected. If the list is longer than one line in length, it may be continued on the next line provided that the last nonblank character of the line is a (leading) delimiter.

After entrance to the PRINT subsystem, a series of questions are asked of the terminal user. Responses to CARD FORMAT? are:

MOVE - implies line numbers are present and are to be moved and printed

STRIP - implies line numbers are present and are not to be printed.

ASIS - implies line numbers are not present in the file.

NORM - implies MOVE option and the standard set of tab characters --

:,8,16,32,73.

If the response was not NORM, the question TAB CHARACTER AND SETTINGS? is asked. Responses are NORM or a series of tab characters and settings of the form:

```
tab ,setting ,setting ...;tab ,setting ,setting ...
```

- LIST

1) LIST

   List the current file on the terminal.

2) LIST i,j

   List all lines of the current file whose line numbers are greater than or equal to i and less than or equal to j. In the case of concatenated files where no sort or resequence has been performed, multiple sets of lines numbered between i and j may or may not be listed if such exist. Either i or j may be omitted. Line numbers 1 or 99999999 respectively will be assumed. If j is omitted, the comma may also be omitted.

3) LIST filedescr (permissions and alt-name are applicable.)

   List the file specified by filedescr on the terminal, without altering the current file. Filedescr must include at least one alpha character if it consists of filename only.

4) LIST filedescr(i,j)$_1$;...;filedescr(i,j,)$_n$ (Permissions and alt-name are applicable.)

   Adjoin and list the specified files or file-segments on the terminal. The current file is not altered. The current file may be included in the list under the name *. If the list is greater than one line in length, it may be continued on the next line provided the last nonblank character on the first line is a (leading) delimiter.

5) LISTH

   List the file with a header (date and time) printed at the top of the listing. LIST formats (1), (2), (3), or (4) may all use the LISTH form instead of LIST.

6) LISTEnnn

   No intervening blanks allowed. List all lines to be "broken" or "folded" at the character position (nnn) specified. Listing of the line will be continued on succeeding line(s). If nnn is omitted, the value 72 is assumed. LIST formats (2) through (4) may also use the LISTEnnn form. Files containing overlength lines (records) may be listed in this manner.

7) LISTS n ,n ,n ,...,n

   List only the specified line(s) n from the current file.

8) LIST 99999999

   If LIST is given with a line number greater than the last line number on the current file, then the last line number of the current file will be printed.

- RESEQUENCE

1) RESEquence

   The line numbers of the current file are resequenced. The resequencing begins with line number 10 and continues in increments of 10. If BASIC is the selected system, the file is resequenced and statement number references in the program are modified correspondingly (GOTO, GOSUB, IF). If FORTRAN or CARDIN was selected, statement number references are not affected.

2) RESEquence n,m

   The line numbers of the current file are resequenced and modifications made according to the system selection. The resequencing begins with line number n and continues in increments of m.

- DELETE

1) DELEte a-b,c-d

   All lines numbered $\underline{a}$ through $\underline{b}$ and $\underline{c}$ through $\underline{d}$ are deleted from the current file.

2) DELEte a

   The line(s) numbered a is deleted from the current file.

3) DELEte a,b,c,d,...

   The lines numbered a,b,c,d, etc., are deleted from the current file.

4) DELEte a,b,c-d,e,f-g,...

   The lines numbered a,b,c through d,e, and f through g are deleted from the current file.

- EDIT

  The Text EDITOR is called into use. Following the READY message, the user may exercise any of the text-editing capabilities available in the Text-EDITOR system. The current file is the recipient of any modification.

- SCAN filedescr (permissions and alt-name are applicable)

  The SCAN subsystem -- batch-output scanner -- is initiated to scan the file described by filedescr. The desired functions are defined by the question/answer sequence that follows the use of this command.

- AUTOMATIC

  1) AUTOmatic

     Causes the automatic creation of line numbers, by the system, at the point at which the automatic mode is entered (or re-entered), with line numbers initially starting at 010 and incrementing by 10 (or, on re-entry, resuming where the previous automatic numbering left off). These line numbers appear in the terminal copy, and are written in the file, just as though the user had typed them.

  2) AUTOmatic n,m

     Causes the automatic creation of line numbers, as above, but starting with line number n and incrementing by m.

Normally the line number will be followed by a blank. Any nonblank, nonnumeric character affixed to the end of the command AUTOmatic will cause the blank to be suppressed. For example: AUTONB or AUTOMATICX.

No commands are recognized while in the automatic mode. The automatic mode is cancelled by giving a carriage return immediately following the issuance of an asterisk and line number by the system.

- STATUS

  List the user's status as to processor time used, number of file I/O's, and characters output to the terminal, and list the files that are open.

  STATUS is also recognized at system-selection level;i.e., it may be typed in response to SYSTEM?

- DONE

  Exit from the selected system to make a new system selection.

- BYE

  Causes the computation of the user's system-usage charges during the session and disconnection of the terminal.

  Depending upon the selected system, the AFT may first be scanned for user's temporary files. A message is issued as to the number of temporary files, then the user is queried as to the disposition. Each filename is printed followed by a question mark. The user may respond as follows:

1) carriage return - implies the file is to be released; pass to next file.

2) NONE - implies all of the succeeding files are to be released.

3) SAVE filedescr - specifies that the file is to be saved on the permanent file described by filedescr. (See PERM command).

BYE may also be given at the system-selection level.

- NEWUSER

Causes the computation of the user's system-usage charges during the session and initiates a new log-on sequence.

NEWUSER may also be given at the system-selection level.

Applicability of Commands by System

	System			
	BASIC	FORTRAN	CARDIN	Text EDITOR
Command				
NEW	yes	yes	yes	no
TAPE	yes	yes	yes	no
#TAPE	no	no	no	yes
OLD	yes	yes	yes	no
LIB	yes	yes	yes	no
SAVE	yes	yes	yes	yes*
PURGE	yes	yes	yes	yes*
REMOVE	yes	yes	yes	no
PERM	no	yes	no	no
GET	yes	yes	yes	no
RUN	yes	yes	yes	no
CATALOG	yes	yes	yes	no
PRINT	no	no	yes	no
LIST	yes	yes	yes	no
RESEQUENCE	yes	yes	yes	no
DELETE	yes	yes	yes	no

EDIT	yes	yes	yes	no
SCAN	no	no	yes	no
AUTOMATIC	yes	yes	yes	no
STATUS	yes	yes	yes	no
DONE	yes	yes	yes	yes*
NEWUSER	yes	yes	yes	no
BYE	yes	yes	yes	no

* "command" in direct-mode.

Appendix B. The GECOS File System and ACCESS

This appendix describes the function and use of the 600TSS ACCESS subsystem in relationship to the GECOS File System. If the user has referenced files only with the SAVE and OLD commands, he does not have to be concerned with ACCESS.

THE GECOS FILE SYSTEM AND ACCESS

The GE-600 Time-Sharing System utilizes the capabilities of the GECOS file system which is a logical mechanism for storing and retrieving permanent files and is common to all programs operating under the GE-600 Comprehensive Operating Supervisor. Since a file system can store many files on some external, "background" storage device, the user need not be concerned with the device his file is on nor the characteristics of the device.

Structure of the File System

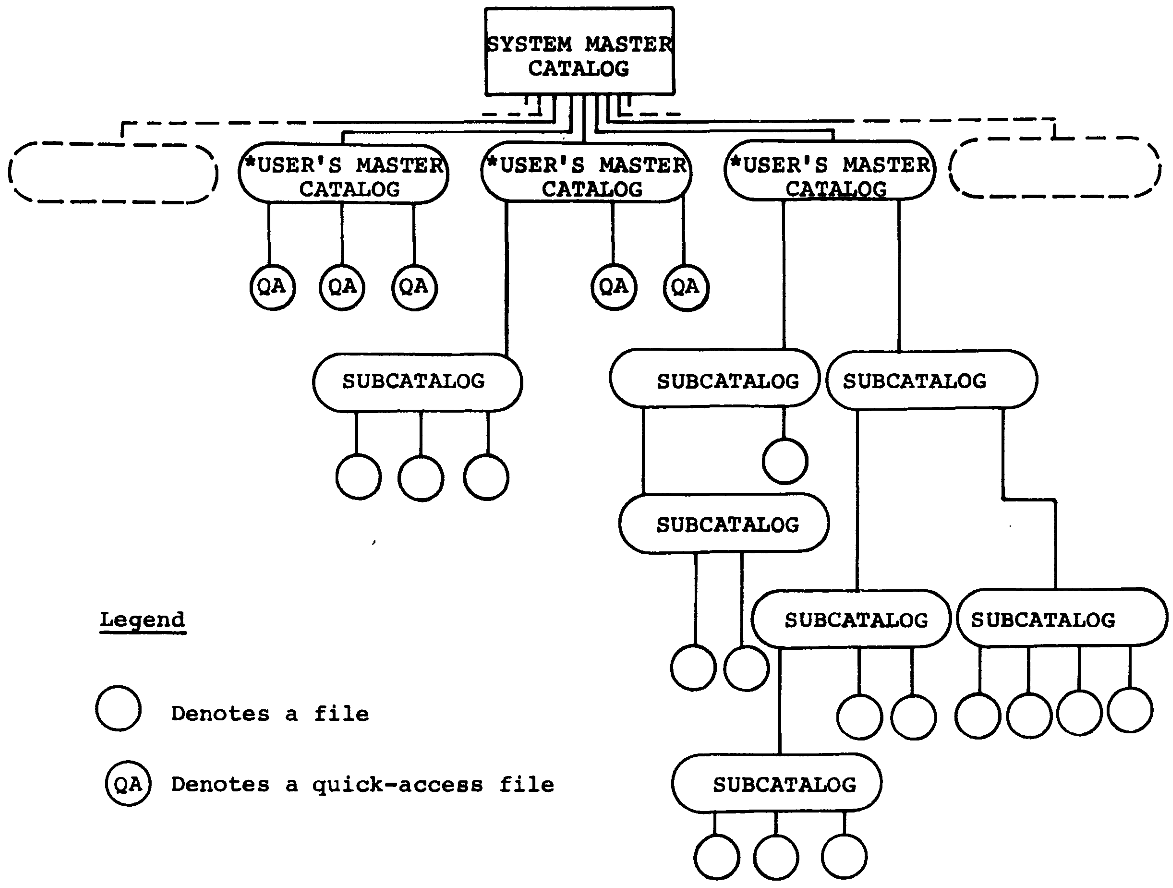
The GECOS file system is described in detail in the GECOS File System manual, CPB-1513. However, the main features of interest to the user will be repeated here.

The GECOS file system represents a tree structure of indefinite length whose origin is the system master catalog. The primary nodes of the tree are user's master catalogs; the lower-level nodes are subcatalogs created by the user. The terminal points of the structure are the files themselves. Figure 1 shows the file system's hierarchical structure.

Catalogs and Files

A catalog consists of a definition containing a catalog name, password, and permissions. Since it contains no user data, a catalog can be neither read nor written, but it is constructed and maintained by the file system itself. The ACCESS utility routine is provided, however, to make catalog changes desired by a user.

A file known to the GECOS file system consists of a definition containing file name, file size, password, permissions, and a description of the physical file space. The file definition is distinct from the physical file space which may contain user data and can be read or written.



All user-ID's must be unique within the system; all subcatalog and file names are automatically qualified by the user's master catalog name and the names of any intermediate subcatalogs. The system master catalog cannot be accessed by the normal user.

---

*Identified by the user-ID.

Figure 1.

Logical Structure of the File System

Passwords

Passwords may be attached to any catalog or file. A password simply allows a user to traverse a catalog/file string. A user can get to a given catalog or file only if he can give the passwords for all higher-level catalogs in the string. (When traversing a string, a password must not be given if none has been attached.) The originator of a given string is required to give the necessary passwords when traversing a string.

Permissions

Users permissions, both general and specific, can be attached to any catalog or file. When permissions are attached at the catalog level, they apply to all subordinate catalogs and files. The originator of a catalog/file string implicitly has all permissions for that string but must give all applicable passwords.

The allowable permissions are:

READ	- allows a file to be read
WRITE	- allows a file to be written
APPEND	- (presently treated as WRITE)
EXECUTE	- (presently treated as READ)
PURGE	- allows catalogs and/or files to be purged from the system, but only with specific permission
MODIFY	- allows catalog and/or file definitions to be changed, but only with specific permission

SAVING AND RETRIEVING FILES IN 600TSS

When operating under the Time-Sharing System, each updated copy of an OLD file or NEW file operated on by BASIC or EDITOR is a temporary working file that will "disappear" at the end of a user's session at the terminal, unless he saves it. This temporary working file, or "scratch copy," allows updates to be made and tested without destroying the original OLD file.

Files are stored with the command SAVE followed by a file name. If the named file has not been previously created, the system automatically creates a permanent, external file and writes the contents of the working file onto it. These files are referred to as quick-access. This means that the file emanates directly from a user's master catalog without intervening subcatalogs (see Figure 1.). Files created by the command SAVE have, by default, general read permission, and are protected against any other form of access, such as write, append or purge.

If the file specified by the SAVE command has been created previously, either by a prior SAVE or by the ACCESS subsystem, the content of the working file is written onto it. This file, if created through ACCESS, is not necessarily of the quick-access type.

The creator of a quick-access file can retrieve it by using OLD and the file name. Therefore, for the user whose needs are met by the characteristics of quick-access files, TSS provides a simple means for utilizing the file system.

ACCESS

Capabilities of the ACCESS Subsystem

For users who wish to utilize some or all of the capabilities of the file system, the ACCESS subsystem provides the interface thus allowing the user to:

- Create hierarchical structures of subcatalogs and files
- Attach passwords to his subcatalogs and files
- Give general permission to all other users to access his files in specified ways
- Give specific permissions, by user-ID
- Protect a given file or set of files against any mode of access
- Gain the permitted types of access to other TSS user's files

- Gain the permitted types of access to files created in the batch-processing environment
- Modify catalog name, password, and/or permissions on an existing catalog
- Modify file name, size, password, and/or permissions on an existing file
- Purge an existing file or catalog/file string
- List all of the catalogs and files which emanate from a given catalog
- Rename files temporarily for a given job

HOW TO USE ACCESS

ACCESS consists of ten functions, which together provide a conversational facility for:

- Creating and purging catalogs and files
- Modifying catalog and file attributes (name, size, password, permissions)
- Accessing and deaccessing files
- Listing catalogs and files

The operation of ACCESS consists of responses, via the terminal, to a sequence of English-language questions. Each function is characterized by a given sequence of questions. All of the "standard vocabulary" associated with the user's responses may be abbreviated for convenience in keying-in.

It should be remembered that ACCESS is not a means of reading or writing permanent file content. OLD and SAVE perform these functions. ACCESS is selected before proceeding to a desired processing subsystem; it is used to "create" or "access" the file, i.e., the file definition and the file space, before the substantive file is built or modified under another subsystem. The OLD/NEW sequence and the SAVE command of the succeeding subsystem(s) are still applicable.

Several general points need to be made in connection with the use of ACCESS:

(1) The definition of a particular catalog or file must include the names of all higher-level catalogs that must be traversed to arrive at that point. The catalog string would include at least the user's master catalog. A file definition, then, is the complete catalog string plus the file name.

It would be inconvenient to give the full file definition each time a file were referred to, so the processing subsystem through the use of the commands OLD and SAVE requires a reference by file name only. This file name is usually the actual one, i.e., the same file name that terminates the full definition. However, since this actual file name might not be unique for all files to be used in one session at the terminal, it is necessary that an alternate file name be supplied for unusual situations. The ACCESS specification of an alternate file name does not change the file definition; it is a local and temporary renaming for the duration of the user's session at the terminal.

(2) Each user's master catalog must be created for him before he can use the system. It has no password or permissions associated with it, and is unalterable. The installation usually controls the generation of this catalog.

(3) The Time-Sharing System maintains an available file table (AFT) for each user. This table is a list, by file name, of the files that the user is going to use during a session at his terminal. All files to be referenced by the SAVE and OLD must either have been placed in AFT by using ACCESS, or be of the quick-access type. When quick-access files are referenced by SAVE or OLD, they are placed in the AFT automatically by the system.

(4) Specific permissions replace general permissions; they do not add to them. That is, if all time-sharing users are given READ permission (general), and users JDOE and FSMITH are given specific WRITE and MODIFY permissions <u>only</u>, for a given file, JDOE and FSMITH cannot read the file. Therefore, in assigning specific permissions, the assignor must specify all permissions granted to the user.

Functions

The initial communication from ACCESS, following system selection, is a request for a choice of function, i.e., FUNCTION?.

The functions that may be requested are:

	(Abbrv.)
● CREATE CATALOG	CC
● CREATE FILE	CF
● ACCESS FILE	AF
● DEACCESS FILE	DF
● MODIFY CATALOG	MC
● MODIFY FILE	MF
● PURGE CATALOG	PC
● PURGE FILE	PF
● LIST CATALOG	LC
● LIST SPECIFIC	LS

The effect produced by each function is as follows:

CREATE CATALOG	- this function creates a subcatalog.
CREATE FILE	- this function defines file space and attributes for a given file name. It does not bring a file into the available file table.
ACCESS FILE	- this function brings a file into the available file table.
DEACCESS FILE	- this function takes a file out of the available file table.
MODIFY CATALOG	- this function modifies the name, password, and/or permissions associated with a given catalog.

MODIFY FILE	- this function modifies the name, size, password, and/or permissions associated with a given file.
PURGE FILE	- this function deletes a file from the system.
PURGE CATALOG	- this function deletes a catalog from the system along with any catalogs and files which are subordinate to it.
LIST CATALOG	- lists the names of the catalogs and files which emanate from this catalog.
LIST SPECIFIC	- lists in detail the description of the catalog or file specified.

Following the response to FUNCTION, ACCESS asks the user to describe the catalog-string, catalog, or file. Each function has a fixed set of questions with several of the questions common to each set. Some of the questions do not logically require a response, e.g., PASSWORD? (there may be none). If no response is applicable, only a carriage return is given.

All the functions, except DEACCESS FILE, first request a definition of the existing catalog-string. Then the name of the catalog or file to be processed is next, along with size attributes in the case of a file. Passwords and permissions are then requested, as appropriate.

Questions and Responses

The sets of questions associated with each function follow, along with the general form of the response to each question. The minimum required response, if any, is underlined. Each set is followed by illustrative examples.

(1) FUNCTION? CREATE CATALOG $\textcircled{\text{CR}}$

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name$password/.../cat-name$password $\textcircled{\text{CR}}$

NEW CATALOG? cat-name $\textcircled{\text{CR}}$

PASSWORD? password (CR)

GENERAL PERMISSIONS? access-type,...,access-type (CR)

The access types are:

READ	(or R)	
WRITE	(or W)	
APPEND	(or A)	
EXECUTE	(or E)	
MODIFY	(or M)	(Specific permission only)
PURGE	(or P)	(Specific permission only)

SPECIFIC PERMISSIONS?

access-type,...,access-type/user-ID/.../user-ID (CR)

The access types are the same as for general permissions.

NOTE: If no response to the question SPECIFIC PERMISSION? is given, i.e., only a carriage-return, the catalog is created and the question NEW CATALOG? is reissued.

Example Replies (user responses are underlined):

FUNCTION? CREATE CATATOG (CR)

CATALOG STRUCTURE TO WORKING LEVEL?

```
JDOE/CAT1$ABC (CR)
```

This response says that there is a subcatalog named CAT1 that is concatenated directly to the user's master catalog identified by the user-ID JDOE, and that it is desired to create a new catalog from this level. The password ABC was attached to catalog CAT1 when it was created.

```
NEW CATALOG?  CAT2 (CR)
```

This response indicates the name of the catalog created at this point.

```
PASSWORD?  AOK (CR)
```

This response associates the password AOK with this catalog. A carriage-return alone would indicate that no password is to be assigned.

```
GENERAL PERMISSIONS? (CR)
```

The lack of a response here indicates that general permission is not granted at this level for any type of access to subsumed files. A response of READ, EXECUTE indicates that any unspecified user has permission to read and execute (if meaningful) any file that emanates from this catalog.

```
SPECIFIC PERMISSIONS?  READ/BJONES/ASMITH (CR)

SPECIFIC PERMISSIONS?  READ,WRITE,PURGE/ALLONG (CR)
```

This combination of responses says that the users who have logged onto the TSS system under the names BJONES and ASMITH can pass through this level with read permission for any files below, and that the user ALLONG can pass through with read, write, and purge permissions.

SPECIFIC PERMISSION? (CR)

The carriage-return alone means that no further specific permissions are to be given; the catalog is now created and the question:

NEW CATALOG?

is reissued, allowing the user to create another catalog at the same level, i.e., also emanating from CAT1.

Alternative forms of the response to CATALOG STRUCTURE TO WORKING LEVEL? are as follows:

`/CAT1$ABC` (CR)

Assuming the user to be JDOE, this response is equivalent to the one given above, JDOE/CAT1$ABC. The initial slant indicates the user's own master catalog.

A response of simply a slant, i.e.:

/ (CR)

indicates that the user desires to create directly from his master catalog. This response is equivalent to his user-ID alone.

(2) FUNCTION? CREATE FILE (CR)

CATALOG STRUCTURE TO WORKING LEVEL?

`user-ID/cat-name$password/.../cat-name$password` (CR)

FILE NAME, SIZE, MAX SIZE?

file name, initial size (links), maximum size (links) (CR)

PASSWORD? password (CR)

GENERAL PERMISSIONS? access-types,...,access-type (CR)

The access-types are:

READ	(or R)	
WRITE	(or W)	
APPEND	(or A)	
EXECUTE	(or E)	
MODIFY	(or M)	(Specific permission only)
PURGE	(or P)	(Specific permission only)

SPECIFIC PERMISSIONS?

access-type,...,access-type/user-ID.../user-ID (CR)

Example Replies (responses are underlined):

FUNCTION? CF (CR)

CATALOG STRUCTURE TO WORKING LEVEL?

/CAT1$ABC/CAT2$AOK (CR)

This response defines user-ID/CAT1/CAT2 as the catalog-string from which the file is to emanate. The initial slant indicates that the succeeding string is concatenated to the user's own master catalog.

FILE NAME, SIZE, MAX SIZE? FIL1,1,3 (CR)

This response asks for a file space of 1 link, initially, with a maximum eventual size limit of 3 links, named FIL1.

PASSWORD? CR

No password is assigned to this individual file.

GENERAL PERMISSIONS? READ CR

SPECIFIC PERMISSIONS? CR

None are granted at this level, but the ones granted at the level of CAT2 (CREATE CATALOG in the previous example), apply to this file.

The lack of a response means the end of the information relevant to the creation of this file. The file is created, and the question

FILE NAME, SIZE, MAX SIZE?

is reissued. This permits creation of other files at the same level.

(3) FUNCTION? ACCESS FILE CR

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name$password/.../cat-name$password CR

FILE NAME$PASSWORD? file name (alternate name) $password CR

PERMISSIONS DESIRED?

access-type,...,access-type CR

The access types are:

- READ (or R)
- WRITE (or W)
- APPEND (or A)
- EXECUTE (or E)

Example Replies (responses are underlined):

FUNCTION? <u>ACCESS FILE</u> (CR)

CATALOG STRUCTURE TO WORKING LEVEL?

<u>JDOE/CAT1$ABC/CAT2$AOK</u> (CR)

The user in this case is not the creator of the file to be accessed, so he must define the user's master catalog (e.g., JDOE) from which the file emanates, along with any required subcatalogs and password.

FILE NAME$PASSWORD? <u>FIL1</u> (CR)

If a password were required, it would be concatenated to the name with a dollar-sign ($), i.e., FIL1$ABC.

PERMISSIONS DESIRED? <u>READ</u> (CR)

General read permissions was granted for this file. (Several specific read permissions were also granted at the level immediately above CAT2). Termination of this response with only a carriage return causes the file to be accessed and the request:

FILENAME$PASSWORD?

to be reissued.

(4) FUNCTION? DEACCESS FILE (CR)

FILE NAME? file name (or CLEARFILES) (CR)

The response for this function is the name of the file to be deaccessed. The name supplied is always the name under which the file was accessed, whether this was the actual name or a temporary alternate name. If CLEARFILES is used, all of the user's available files are deaccessed including his temporary files.

(5) FUNCTION? PURGE CATALOG (CR)

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name$password/.../cat-name$password (CR)

CAT. TO BE PURGED? cat-name (CR)

PASSWORD? password (CR)

Example Replies (responses are underlined):

FUNCTION? PC (CR)

CATALOG STRUCTURE TO WORKING LEVEL?

/CAT$ABC (CR)

This response defines the subcatalog CAT1 concatenated to the user's own master catalog.

CAT. TO BE PURGED? CAT2 (CR)

PASSWORD? AOK (CR)

The dollar-sign is used only when the password is concatenated directly to a file or catalog name. The request

CAT. TO BE PURGED?

is reissued.

(6) FUNCTION? PURGE FILE (CR)

CATALOG STRUCTURE TO WORKING LEVEL?

user-ID/cat-name$password/.../cat-name$password (CR)

FILE TO BE PURGED? file name (CR)

PASSWORD? password (CR)

Example Replies (responses are underlined):

FUNCTION? PF (CR)

CATALOG STRUCTURE TO WORKING LEVEL?

JDOE /CAT1$ABC/CAT2$AOK (CR)

The user in this case is ALLONG, not the file creator.

FILE TO BE PURGED? FIL1 (CR)

PASSWORD? (CR)

The user (ALLONG) was given specific purge permission at the level of CAT2.

The request

FILE TO BE PURGED?

is reissued.

(7) FUNCTION? MODIFY CATALOG (CR)

CATALOG STRUCTURE INCLUDING CATALOG TO BE MODIFIED?

user-ID/cat-name$password,...,cat-name$password (CR)

NEW NAME? new cat-name (CR)

PASSWORD? $\left\{\begin{array}{l}\text{new password}\\ \text{DELETE}\end{array}\right\}$ (CR)

GENERAL PERMISSIONS? $\left\{\begin{array}{l}\text{access-type,...,access-type}\\ \text{DELETE}\end{array}\right\}$ (CR)

SPECIFIC PERMISSIONS? $\left\{\begin{array}{l}\text{access-type,...,access-type/}\\ \text{user-ID.../user-ID}\\ \text{DELETE/user-ID/.../user-ID}\end{array}\right\}$ (CR)

Example Replies (user responses are underlined):

FUNCTION? <u>MC</u> (CR)

CATALOG STRUCTURE INCLUDING CATALOG TO BE MODIFIED?

<u>/CAT1$ABC/CAT2$AOK</u> (CR)

NEW NAME? (CR)

A carriage-return only response means that the catalog name is to remain unchanged.

PASSWORD? <u>XYZ</u> (CR)

The original password AOK is replaced by XYZ.

GENERAL PERMISSIONS? <u>READ</u> (CR)

As originally created, general permissions were not assigned at this level. This reponse replaces this null set with READ permission.

SPECIFIC PERMISSIONS? <u>R,W/BJONES</u> (CR)

This response replaces the original specific READ permission for BJONES with READ and WRITE permission.

SPECIFIC PERMISSIONS? <u>DELETE/ASMITH</u> (CR)

This response cancels any permissions for ASMITH that previously existed.

SPECIFIC PERMISSIONS? R,W,P,M/ALLONG (CR)

This response replaces the original set of READ, WRITE and PURGE permissions for ALLONG with READ, WRITE, PURGE, and MODIFY.

SPECIFIC PERMISSIONS? (CR)

The carriage-return above implies that no further modifications are to be made; the changes are now processed and the question:

CATALOG STRUCTURE INCLUDING CATALOG TO BE MODIFIED?

is reissued.

(8) FUNCTION? MODIFY FILE (CR)

CATALOG STRUCTURE INCLUDING FILE TO BE MODIFIED?

user-ID/cat-name$password/.../
cat-name$password/file-name$password (CR)

NEW NAME? new file name (CR)

NEW MAX SIZE? new maximum size (in links) (CR)

PASSWORD? $\left\{\begin{array}{l}\text{new password}\\ \text{DELETE}\end{array}\right\}$ (CR)

GENERAL PERMISSIONS? $\left\{\begin{array}{l}\text{access-type,...,access-type}\\ \text{DELETE}\end{array}\right\}$ (CR)

$$\text{SPECIFIC PERMISSIONS?}\ \left\{\begin{array}{l}\text{access-type/user-ID/.../user-ID}\\ \text{DELETE/user-ID/.../user-ID}\end{array}\right\}\text{(CR)}$$

Example Replies (responses are underlined):

FUNCTIONS? $\underline{\text{MF}}$ (CR)

CATALOG STRUCTURE INCLUDING FILE TO BE MODIFIED?

$\underline{\text{/CAT1\$ABC/CAT2\$XYZ/FIL1}}$ (CR)

NEW NAME? $\underline{\text{MASTER1}}$ (CR)

NEW MAX SIZE? $\underline{\text{5}}$ (CR)

This response increases the maximum file size to 5 links (originally 3).

PASSWORD? $\underline{\text{DEPT37}}$ (CR)

This response attaches the password DEPT37 to this file (none originally assigned).

GENERAL PERMISSIONS? $\underline{\text{DELETE}}$ (CR)

The original general READ permission is deleted.

SPECIFIC PERMISSIONS? $\underline{\text{P/BJONES}}$ (CR)

PURGE permissions for user BJONES is added at this level. This permission applies to this file only, but he also has READ and WRITE from the CAT2 level.

(9) FUNCTION? LIST CATALOG (CR)

CATALOG STRUCTURE INCLUDING CATALOG TO BE LISTED?

user-ID/cat-name,...,cat-name (CR)

Example Replies (user responses are underlined):

FUNCTION? LC (CR)

CATALOG STRUCTURE INCLUDING CATALOG TO BE LISTED?
/CAT1

Passwords need not be given in the catalog structure. A user is permitted to list only his own catalogs on the LIBRARY catalog.

A list of the catalogs and files emanating from CAT1 would now be output.

(10) FUNCTION? LIST SPECIFIC (CR)

CATALOG STRUCTURE INCLUDING CATALOG OR FILE TO BE LISTED?

user-ID/cat-name,...,cat-name(or)file-name (CR)

Example Replies (user responses are underlined):

FUNCTION? LS (CR)

CATALOG STRUCTURE INCLUDING CATALOG OR FILE TO BE LISTED?

/CAT1

Passwords need not be given in the catalog structure and will not be included in the catalog or file description which is output. A user can list only his own catalogs or files.

The description of CAT1 would now be output.

Identifiers and Delimiters in User Responses

User responses are composed of the following:

- Identifiers
- Keywords
- Word delimiters
- Line delimiters

Identifiers consist of file names, catalog names, user-IDs and passwords. They can consist of alphabetics, numerics, periods, and minus signs. Each identifier can be up to 12 characters in length except file names which are limited in length to 8 characters.

In response to the question FILE NAME$PASSWORD?, issued by the Access File function, a file name of up to 12 characters may be specified, i.e., the name of a batch-environment file, if followed by an alternate name of 8 characters or less, enclosed in parentheses. Also, in response to FILE TO BE PURGED?, a file name of up to 12 characters could be specified, if the file to be purged were not created in the TSS environment.

Keywords consist of function names, access types (permissions), and several file-type parameters, of limited interest, that are described under "Special Features." Keywords are used in responses to questions, and can always be abbreviated to the initial character, or a two-character acronym in the case of function name, e.g., R for READ permission or CC for CREATE CATALOG function.

The file-size specification in the response to FILE NAME, SIZE, MAX SIZE? (Create File), is a decimal number denoting the number of links required. This may be considered a special case of keyword.

Word delimiters are used in user responses: the slant or virgule (/), the dollar-sign ($), and the comma (,). Blanks may be used freely in responses except within function names; they are in no sense delimiters and are ignored.

The use of the three delimiters is as follows:

The / delimiter has two functions:

(1) In catalog-strings a slant indicates that a subcatalog name follows and is concatenated to the preceding catalog in the string. An initial slant indicates that the following subcatalog-string (if any) is concatenated to the user's master catalog. A response to CATALOG STRUCTURE TO WORKING LEVEL? of / (CR) is equivalent to the user's own user-ID, i.e., it positions the user to his own master catalog.

(2) In specific permissions a slant indicates that a user-ID follows.

The $ is used only to concatenate a password to a catalog or file name.

The , is used as a general separator for keywords, i.e., for separating access-types and sizes, and separating file names from the following keywords or sizes.

The line delimiters are a carriage-return, an asterisk (*) plus carriage-return, or a double asterisk (**) plus a carriage-return. Each of these serves to terminate a response, but with a different effect.

(1) Carriage Return: A carriage-return following a response generally signifies that the user wishes to remain at the same catalog position (if relevant), and proceed to the next question in logical sequence. This may be the next question in a set, or the initial question again.

When only a carriage-return is given, however, i.e., a "null" response, it has several possible meanings:

- In response to the question CATALOG STRUCTURE TO WORKING LEVEL? a carriage-return-only is equivalent to the user's own user-ID or a slant, i.e., / (CR). Any of these responses requests that the user be positioned to his own master catalog.

- A carriage-return-only following a question that logically requires a response, e.g., NEW CATALOG?, causes an immediate return to the question FUNCTION?.

- The question SPECIFIC PERMISSIONS? recurs each time a response is given (delimited by a carriage-return), since only one set of specific permissions can be given in each. If only a carriage-return is given, the information received so far is processed, and the first question below CATALOG STRUCTURE TO WORKING LEVEL? is reissued, i.e., NEW CATALOG? or FILE NAME, SIZE, MAX SIZE?, allowing a new catalog or file to be created at the same catalog level.

- A carriage-return only response to FUNCTION? returns the question SYSTEM?.

(2) Single Asterisk Plus Carriage-Return: If a single asterisk plus a carriage-return is given in reply to a question, either with or without a substantive response, ACCESS processes the information it has and returns to the first question at the same catalog level, e.g., to skip any further questions in the set.

(3) Double Asterisk Plus Carriage-Return: If a double asterisk plus a carriage-return is given, either with or without a substantive response, ACCESS processes the information it has and returns to the question "FUNCTION?". It implies that the user is finished with the current function.

In addition to the changes in level of operation produced by the several line delimiters, a response of "DONE" to any question causes an exit from ACCESS. No processing is performed and the question "SYSTEM?" results.

The line delimiters show that there are several ways of either shortening the question-response sequence, or terminating a function at any given point.

Examples of the effect of different response terminations:

FUNCTION? $\underline{\text{CC}}$ $\textcircled{\text{CR}}$

CATALOG STRUCTURE TO WORKING LEVEL?

$\textcircled{\text{CR}}$ (The carriage-return only implies master catalog.)

NEW CATALOG? $\underline{\text{001*}}$ $\textcircled{\text{CR}}$

(This indicates that passwords or permissions are not wanted for this catalog and no further questions are wanted.)

NEW CATALOG? $\underline{\text{002}}$ $\textcircled{\text{CR}}$

PASSWORD? $\underline{\text{PASS2**}}$ $\textcircled{\text{CR}}$

(This implies that no permissions are to be assigned to this catalog, and that creating catalogs at this position is finished.)

FUNCTION? $\underline{\text{CF}}$ $\textcircled{\text{CR}}$

CATALOG STRUCTURE TO WORKING LEVEL?

$\underline{\text{/002\$PASS2}}$ $\textcircled{\text{CR}}$

FILE NAME, SIZE, MAX SIZE? $\underline{\text{02.1,1,3}}$ $\textcircled{\text{CR}}$

GENERAL PERMISSIONS? $\underline{\text{READ}}$ $\textcircled{\text{CR}}$

SPECIFIC PERMISSIONS? $\underline{\text{W/RJJONES**}}$ $\textcircled{\text{CR}}$

(This implies that creating files at this level has been completed.)

```
    FUNCTION? (CR) (or DONE (CR))

        (finished with ACCESS)

    SYSTEM?
```

General Usage Rules

The ability of a user to access files and otherwise manipulate catalog/file structures, e.g., modifying and purging, depends upon his knowing the necessary file definitions. Beyond this, the file system has two file and catalog protection features: passwords and permissions.

Permissions provide the file creator a positive protection feature; if permissions are not explicity granted, his catalogs and files are completely protected by default. The user must assign to others any degree of access he wishes them to have. But, note that since specific permissions for a given user do not add to, but replace, any general permission that may have been given, specific permissions may be used to exclude a given set of users from one or more types of access.

Passwords provide an additional level of protection. If passwords are assigned by the creator of a catalog/file string they must be supplied in order to pass through the string.

The creator of a catalog/file string is exempt from any access-mode restrictions he imposes, i.e., he implicitly has all permissions for <u>his</u> catalogs and files, but he must give all passwords.

The MODIFY permission, which allows another user to change file names, catalog names, file size, passwords, and/or permissions, also implies the ability of this other user to create catalogs and/or files emanating from the master catalog.

Special Features

Using the Create-File function, previously described, the files created are not necessarily contiguous; i.e., successive links of a multi-link file are not necessarily in physical sequence on the storage device. Furthermore, both the Create-File and Access-File functions assume that the file will be treated as a linked file. For the standard subsystems provided with 600TSS, these file characteristics are suitable because linked files are required.

If, however, in the use of a given subsystem, it would be advantageous to have contiguous files, this characteristic can be specified in response to FILE NAME, SIZE, MAX SIZE?. The form of this response is:

file name,initial size C

The parameter "C" indicates, in Create File only, that a contiguous file is desired. No maximum size may be specified.

Similarly, if random treatment of files is required in a given user-written subsystem, a file can either be created as a random file or accessed as a random file. If created as such, it is always treated by the GECOS I/O Supervisor as a random file. If it is created as a linked file, it can be accessed as a random file, but in that case, the random treatment indication is temporary, i.e., it applies to that access only.

The forms of the random specification are as follows:

For Create-File, the response to FILE NAME, SIZE, MAX SIZE? is:

file-name,initial size,maximum size,R

(or)

file name,initial sizeC,R

For Access-File, the response to FILE NAME$PASSWORD? is:

file-name,R$password

In both responses, the parameter R (always preceded by a comma) indicates that the named file is to be treated as a random file.

REQUEST DENIED MESSAGES

The following messages are printed following a complete function request, and indicate that the request could not be satisfied. The reason for denial is given in each case.

REQUEST DENIED-NEW NAME SAME AS AN EXISTING NAME

A new catalog or file name has been given that is the same as an existing catalog or file name at the same level.

REQUEST DENIED-FILE SPACE REQUESTED EXCEEDS ALLOWED

The user has requested file space exceeding the amount that has been allotted to him in his System Master Catalog entry.

REQUEST DENIED-NEW SIZE LESS THAN CURRENT SIZE

In MODIFY FILE, a new file size has been specified which is less than that currently used by the file.

REQUEST DENIED-SYSTEM MALFUNCTION

An unrecoverable I/O error has occurred.

REQUEST DENIED-PERMISSION NOT GRANTED

The user does not possess the requested permission(s).

REQUEST DENIED-FILE BUSY

The requested file is currently busy to the type of permission(s) requested.

REQUEST DENIED-INCORRECT CAT/FILE DESCRIPTION

This denial is given whenever required passwords are not included or the catalog/file description is not logically correct.

REQUEST DENIED-SYSTEM LOADED

The requested file function cannot be completed because there is temporarily no file space available.

REQUEST DENIED-YOUR AVAILABLE FILE TABLE IS FULL

The user has too many files accessed (open) at the same time. This situation can be eliminated by deaccessing some of the accessed files.

REQUEST DENIED-FILE NAME A DUPLICATE, MUST GIVE ALTERNATE NAME

An ACCESS FILE has been done where the file name is a duplicate of a file which the user currently has open. The alternate name capability can be used to avoid this situation.

INPUT ERROR MESSAGES

The following messages are printed immediately following the input in error, and the original question is repeated.

ERR-ILLEGAL CHARACTER

A character other than an alphabetic, numeric, period, or dash has been included in an identifier. An upward arrow (↑) points at the character in error.

ERR-INVALID DELIMITER

An otherwise valid delimiter has been given out of place. An upward arrow (↑) points at the delimiter in error.

ERR-XXXXXXXXXXXX-MUST BE LESS THAN 13 CHARACTERS

The designated identifier is limited to 12 characters.

ERR-XXXX-IS NOT A LEGITIMATE PERMISSION

Legitimate permissions are READ, WRITE, APPEND, and EXECUTE, plus PURGE and MODIFY as specific permissions only.

ERR-XXXXXXXXXX-MUST BE LESS THAN 9 CHARACTERS

The designated identifier is limited to 8 characters.

ERR-XXXX-MUST BE ALL NUMERIC

A non-numeric character has been included in field XXXX.

ERR-XXXXX-MUST BE LESS THAN 1000

The field is limited to three digits.

ERR-INPUT REQUIRED

A null response was given to a question which requires input.

ERR-INITIAL SIZE GREATER THAN MAX SIZE

In defining the file size an initial size greater than the maximum size was given.

Appendix C.
Time-Sharing FORTRAN and FORTRAN IV Differences

Time-Sharing FORTRAN language differs from FORTRAN IV in several important ways. In Time-Sharing FORTRAN:

1. Mixed modes are allowed in arithmetic expressions.
2. Quotation marks can be used in format statements in place of a hollerith count.
3. File names are used in place of tape numbers.
4. ASCII and filename constants have been added to the system.
5. A subscript may take the form of any legal FORTRAN arithmetic expression.
6. The following input/output statements are allowed:

   ```
   READ:
   PRINT:
   PRINT: "n" (where n is any hollerith information)
   ```

   Information enclosed in quotation marks may be included in any list of variables associated with an output statement. For example, the following statements are both valid:

   ```
   PRINT:ABC,Q*27.6,"ANSWER=",XYZ
   WRITE(NAME,999)PQR,"CLASS"
   ```

In the first statement, "ANSWER=" is simply hollerith information which will be printed out in front of the answer. In the second statement, "CLASS" is defined as a filename constant and will be written as two words. (See Chapter 4 for a definition of filename constant.)

Character strings within quotation marks used after PRINT: may be any length desired, but in any other usage they are one or two words long, blank-filled. (See Chapter 4 for definition of ASCII and filename constants.)

7. Multiple entry points are not allowed in subroutine or function subprograms.

8. Labeled Common is not allowed.

9. Block data subprograms are not allowed.

10. Complex arithmetic is not allowed.

11. Double-precision arithmetic is not allowed.

12. Equivalence is not allowed at the present time.

13. Arithmetic statement functions and dimensioned variables cannot be named FORMAT.

14. Variable names may not be the same as subprogram or arithmetic statement functions.

Appendix D. Sample Problems

This appendix gives examples of how files are created, edited, saved, and compiled. The examples illustrate the time-sharing system log-on procedure, saving of the file, checkout, and termination of the connection.

PROBLEM A

This example illustrates a method for solving the quadratic equation $ax^2+bx+c=0$. Six different equations are solved, but the number can be varied by changing the test of "COUNT". Each time a new equation is used, the problem calls for new values of a, b, and c to be read in from the terminal.

```
THIS IS THE GE-600 TSS SYSTEM ON 08/01/68 AT 8.553 CHANNEL 0013

USER ID - WLJACKSON
PASSWORD--
XKKEKKEKXXKK
SYSTEM ? FORTRAN
OLD OR NEW-NEW
READY FOR INPUT
*AUTOX
```

```
*0010*      PROGRAM QUAD:  QUADRATIC EQUATION SOLUTION
*0020       COUNT = 0
*0030  10   READ:I,J,K
*0040       COUNT = COUNT + 1
*0050       TEMP = J**2-4*I*K
*0060       IF (TEMP)30,40,40
*0070  40   X1 = (-J+SQRT(TEMP))/(I*2)
*0080       GO TO 90
*0090  30   A = J/(I*2)
*0100       V = SQRT(-TEMP)
*0110       B = V/(I*2)
*0120       PRINT 50,A,B
*0130  50   FORMAT(1H9,"IMAGINARY ROOTS:REAL = ",I3,
*0140&      "IMAG=+ OR -",I2,"I",/)
*0150  80   IF(COUNT -6)10,60,10
*0160  90   PRINT 70,X1,X2
*0170  70   FORMAT(1X,"X1=",I3,"X2=",I3/)
*0180       GO TO 80; 60 STOP
*0190       END
*0200
```

(In checking the listing, the user notes the lack of a means for computing X2. He then inserts the required statement.)

```
*0075       X2=(-J-SQRT(TEMP)/(I*2)

*SAVE QUAD
DATA SAVED--QUAD
```

(With the source program saved, the user compiles and executes the program via the RUN command.)

```
*RUN

=1,6,9
X1=072, X2=072
= 2,6,4
X1-048, X2=504
= 1,-4,-5
X1=488, X2=048
= 2,4,4


IMAGINARY ROOTS:REAL = 728 IMAG=+ OR -201
= 1,2,-15
X1=208, X2=744
= 1,-7,-8
X1=688,X2=048

PROGRAM STOP AT 180
 *BYE
**RESOURCES USED $  1.22, USED TO DATE $  927.43= 93%
**TIME SHARING OFF AT 8.647 ON 08/01/68
```

PROBLEM B

This example shows a program for producing the mean and standard deviation of each of n samples, each consisting of exactly three data-points.

```
THIS IS THE GE-600 TSS SYSTEM ON 10/08/68 AT 13.878 CHANNEL 0013


USER ID -WLJACKSON
PASSWORD--
[redacted]


SYSTEM  ? FORTRAN
OLD OR NEW-NEW
READY FOR INPUT
*AUTOX
*0010*      MEAN AND STANDARD DEVIATION PROBLEM
*0020       DIMENSION A(3),TOTAL(3),SUMSQ(3),COUNT(3)
*0030       DO 10 I=1,3
*0040       TOTL@AL(I)=0.
*0050       SUMSQ(I)=0
*0060       COUNT(I)=0.
*0070       NUM=0
*0080       READ 3,K
*0090  60   IF(NUM-K)20,30,200@
*0100  20   READ 1,A
*0110       NUM=NUM+1
*0120       DO 40 I=1,3
*0130       IF(A(I))40,40,50
*0140  50   TOTAL(I)=TOTAL(I)+A(I)
*0150       SUMSQ(I)=SUMSQ(I)+A(I)**2
*0160       COUNT(I)=COUNT(I)+1
*0170  40   CONTINUE
*0180       GOTO 60
*0190  30   DO 70 I=1,3
*0200       XBAR=TOTAL(I)/COUNT(I)
*0210       STDEV=SQRT((SUMSQ(I)-(TOTAL(I)/COUNT(I))
*0220&**2)/(COUNT(I)-1))
*0230  70   PRINT 2,XBAR,STDEV
*0240  2    FORMAT (5X,2F12.4)
*0250  1    FORMAT(F5.2,2X,F5.2,2X,F5.2)
*0260  3    FORMAT(I2)
*0270       STOP
*0280       END
*0290
```

```
*0060  10  COUNT(I)=0.
*SAVE STDEV
DATA SAVED--STDEV
```

(In looking over the listing, the user sees that statement number 10 to complete the first DO loop is missing. He corrects the error before saving the source file.)

(With the source program saved, the user compiles and executes the program via the RUN command.)

```
*RUN
= 06
= 15.12  61.13   14.07
= 10.30   7.09   11.12
= 18.20  42.31   38.03
=  5.55  24.16   21.03
= 61.43   6.10   15.78
= 27.13  17.71   14.98

         23.1217     30.5674
         26.5833     34.0739
         19.3350     21.4929


PROGRAM STOP AT 270
SYSTEM  ?
```

(The user then typed in all his data. He gave 06 as the value for K, followed by 6 groups of three numbers each. The computer then executed the program, printed the mean and standard deviation values, and returned control to the user at the system selection level.)

Appendix E. Built-In Function Examples

FUNCTION	EXAMPLE

Absolute value	Y=ABS(-100.)
	Y=100.

Truncation	Y=AINT(-100.1)
	Y=-100.

Remaindering	Y=AMOD (5.,2.)
	Y=1.

Choosing largest value	Y=AMAX1(10., 1., 100.,5.)
	Y=100.

Choosing smallest value	Y=AMINO (10., 1., 50.)
	Y=1.

Float	Y=FLOAT(10)
	Y=10.

Fix	I=IFIX(10.)
	I=10

Transfer of Sign	Y=SIGN(10., -1.)
	Y=-10.

Positive difference	Y=DIM(5., 2.,)
	Y=3.

Mathematical Expression	FORTRAN Expression	Range of Arguments	Errors and Error Codes
I^J	I**J		1. I = 0, J = 0 EXPONENTIATION ERROR 0**0 SET RESULT = 0 2. I = 0, J < 0 EXPONENTIATION ERROR 0**(-J) SET RESULT = 0
B^J	B**J		1. B = 0, J = 0 EXPONENTIATION ERROR 0**0 SET RESULT = 0 2. B = 0, J < 0 EXPONENTIATION ERROR 0**(-J) SET RESULT = 0
B^C	B**C		1. B = 0, C = 0 EXPONENTIATION ERROR 0880 SET RESULT = 0 2. B = 0, C < 0 EXPONENTIATION ERROR 088(-C) SET RESULT = 0 3. B < 0, C ≠ 0 EXPONENTIATION ERROR (-B)**C EVALUATE FOR +B

Mathematical Expression	FORTRAN Expression	Range of Arguments	Errors and Error Codes
e^B	EXP(B)	$\lvert B \rvert \leq 88.028$	$\lvert B \rvert > 88.028$ EXP(B), B GRT THAN 88.028 NOT ALLOWED SET RESULT = ARGUMENT
$\log_e B$	ALOG(B)	$B > 0$	1. $B = 0$ LOG(0) NOT ALLOWED SET RESULT = 0 2. $B < 0$ LOG(-B) NOT ALLOWED EVALUATE FOR +B

sinB
cosB
	SIN(B)
	COS(B)
		$B < 2^{27}$	$B \geq 2^{27}$
			SIN OR COS ARG GRT TH 2**27
			NOT ALLOWED
			SET RESULT = 0

$\sqrt{B}$	SQRT(B)	$B \geq 0$	$B < 0$ SQRT(-B) NOT ALLOWED EVALUATE FOR +B

Mathematical Expression	FORTRAN Expression	Range of Arguments	Errors and Error Codes
ARCTANGENT X	ATAN(X) $-\pi/2 \le$ATAN(X)$\le \pi/2$	any argument	
	ATAN2,(X,Y) $-\pi \le$ATAN2(X,Y)$\le \pi$	$(X,Y) \ne (0,0)$	(X,Y) = (0,0) ATAN2(0.0) NOT ALLOWED SET RESULT = 0

ARCSINE
ARCCOSINE
	ARSIN(X)
	$-\pi/2 \le$ARSIN(X)$\le \pi/2$
	ARCOS(X)
	$0 \le$ARCOS(X)$\le \pi$
		$\|A\| \le 1$	$\|A\| > 1$
			|ARG| GREATER THAN 1.0
			EVALUATE FOR ARG = 1

TANGENT
COTANGENT
	TAN(X)
	COTAN(X)
		$\|X\| \le 2^{27}$	1. $X > 2^{27}$
			TAN OR COT ARG GRT THE 2**27
			NOT ALLOWED
			SET RESULT = 0
			2. $X < 2^{-126}$ IN COT ARGUMENT
			OUT OF RANGE
			SET RESULT = INFINITY

erf(X)	ERF(X)	ERF(-X)=-ERF(X)	
Γ(X)	GAMMA(X)	X cannot be 0 or a negative integer	
log (Γ(X))	ALGAMA(X)		

Appendix G. Glossary

ASCII	- An abbreviation for the character code defined by the American Standard Code for Information Interchange.
Binary	- The term used to define a condition which has two states or alternatives; more generally used to refer to words consisting of a number of binary bits. In the TSS FORTRAN system, "binary" is used to specify linked files of records transmitted by means of a list that specifies words rather than fields of characters.
BREAK key	- A nonprinting character which when typed will terminate execution. It is transmitted by typing the key marked BREAK or INTERRUPT on the terminal. Should be used very sparingly, as its effects are often unpredictable, especially with regard to the status of files.
Current File	- A temporary file created by the system to permit a user to create and/or reference, modify and possibly execute a file without permanently saving it. It is the file presently being accessed by the user in the sense that the user creates it either by NEW or by specifying a file by name with an OLD command and may subsequently operate on it without specifying it by name.
Disc or Drum File	- A storage device capable of storing large quantities of information for individual users and the system itself.
EFN	- External formula number; same as statement number.

Link	- A fixed-length area relative to a specific storage device. In the TSS system it is used to refer to 3840-word area on the disc or drum file.
Linked file	- A file made up of links which are not necessarily contiguous but are processed serially.
Password	- An optional code from one to 12 characters, assigned by the user to protect his files against unauthorized access by others.
Random file	- A file processed by direct addressing.
Time-Sharing	- The simultaneous use of a computer by many users.
User	- That to which the system reacts.
User-ID	- A code of from one to 12 characters that is uniquely assigned to each user for purposes of identification with regard to file storage and billing.
User's Master Catalog	- A table with an entry for each file (or subcatalog) of a user. Each entry contains the file (or catalog) name, the number of links used, the date the file was last accessed, the date it was last modified, the mode, permissions, and the password, if any.
Variable	- A symbol which represents a quantity whose value is assigned dynamically elsewhere in the program.
=	- This character (equal sign), when transmitted to the user, requests a reply in the form of data required by the program.

Appendix H. ASCII Character Set

The 7-bit octal codes which require a parity bit of 1 are underlined. Characters in memory do not have parity with them. It is automatically deleted on input and subsequently generated on output. 'C' designates the control key. 'CS' designates the control and shift keys together. The non-printing control characters are (by 7-bit code), Ø16, Ø17, Ø2Ø, Ø22, and Ø24.

7-BIT ASCII		MODEL 33/35		
CODE	CHAR	KEY	CHAR	ASCII INTENT
000	NUL	'CS'P		Null or time fill character
001	SOH	'C'A		Start of Heading
$\overline{0}02$	STX	'C'B		Start of Text
$\overline{0}03$	EXT	'C'C	EOM	End of Text
004	EOT	'C'D		End of Transmission
$\overline{0}05$	ENQ	'C'E	WRU	Enquiry (who are you)
006	ACK	'C'F	RU	Acknowledge
007	BEL	'C'G		Bell
$\overline{0}10$	BS	'C'H		Backspace
$\overline{0}11$	HT	'C'I		Horizontal Tabulation
012	LF	LINE FEED		Line Feed (or New Line)
013	VT	'C'K		Vertical Tabulation
$\overline{0}14$	FF	'C'L		Form Feed
015	CR	RETURN		Carriage Return
$\overline{0}16$	SO	"C'N		Shift Out
$\overline{0}17$	SI	'C'O		Shift In
020	DLE	'C'P		Data Link Escape
$\overline{0}21$	DC	'C'Q	X-ON	Device Control 1
022	DC	'C'R	TAPE-AUX. ON	Device Control 2
023	DC	'C'S	X-OFF	Device Control 3
$\overline{0}24$	DC	'C'T	TAPE-AUX.OFF	Device Control 4
025	NAK	'C'U	ERROR	Negative Acknowledge
$\overline{0}26$	SYN	'C'V		Synchronous Idle
$\overline{0}27$	ETB	'C'W		End of Transmission Blocks
030	CAN	'C'X		Cancel
031	EM	'C'Y		End of Medium
$\overline{0}32$	SS	'C'Z		Special Sequence

033	ESC	'CS'K	Escape
034	FS	'CS'L	File Separator
$\overline{0}$35	GS	'CS'M	Group Separator
036	RS	'CS'N	Record Separator
037	US	'CS'O	Unit Separator
$\overline{0}$40	SP		Space
$\overline{0}$41	!		Exclamation Point
042	"		Quotation Marks
043	#		Number Sign
$\overline{0}$44	$		Currency Symbol
045	%		Percent
$\overline{0}$46	&		Ampersand
$\overline{0}$47	'or´		Apostrophe or Acute Accent
050	(		Opening Parenthesis
051	)		Closing Parenthesis
$\overline{0}$52	*		Asterisk
$\overline{0}$53	+		Plus
054	,		Comma or Cedilla
$\overline{0}$55	-		Hyphen or Minus
056	.		Period or Decimal Point
057	/		Slant
$\overline{0}$60	0		Zero
061	1		One
$\overline{0}$62	2		Two
$\overline{0}$63	3		Three
064	4		Four
$\overline{0}$65	5		Five
066	6		Six
067	7		Seven
$\overline{0}$70	8		Eight
$\overline{0}$71	9		Nine
072	:		Colon
073	;		Semicolon
$\overline{0}$74	<		Less Than
075	=		Equal
$\overline{0}$76	>		Greater Than
$\overline{0}$77	?		Question Mark
100	`	@	Grave Accent
$\overline{1}$01	A		Capital Letter
102	B		Capital Letter
103	C		Capital Letter
$\overline{1}$04	D		Capital Letter
105	E		Capital Letter
$\overline{1}$06	F		Capital Letter
$\overline{1}$07	G		Capital Letter
110	H		Capital Letter
111	I		Capital Letter
$\overline{1}$12	J		Capital Letter
$\overline{1}$13	K		Capital Letter
114	L		Capital Letter
$\overline{1}$15	M		Capital Letter

116	N			Capital Letter
117	O			Capital Letter
120	P			Capital Letter
121	Q			Capital Letter
122	R			Capital Letter
123	S			Capital Letter
124	T			Capital Letter
125	U			Capital Letter
126	V			Capital Letter
127	W			Capital Letter
130	X			Capital Letter
131	Y			Capital Letter
132	Z			Capital Letter
133	[	'S'K		Opening Bracket
134	~	'S'L	\	Tilde
135	]	'S'M		Closing Bracket
136	^		↑	Circumflex
137	_		←	Underline
140	@			Commercial "At"
141	a			Lower case alphabetics
142	b			Lower case alphabetics
143	c			Lower case alphabetics
144	d			Lower case alphabetics
145	e			Lower case alphabetics
146	f			Lower case alphabetics
147	g			Lower case alphabetics
150	h			Lower case alphabetics
151	i			Lower case alphabetics
152	j			Lower case alphabetics
153	k			Lower case alphabetics
154	l			Lower case alphabetics
155	m			Lower case alphabetics
156	n			Lower case alphabetics
157	o			Lower case alphabetics
160	p			Lower case alphabetics
161	q			Lower case alphabetics
162	r			Lower case alphabetics
163	s			Lower case alphabetics
164	t			lower case alphabetics
165	u			Lower case alphabetics
166	v			Lower case alphabetics
167	w			Lower case alphabetics
170	x			Lower case alphabetics
171	y			Lower case alphabetics
172	z			Lower case alphabetics
173	{			Opening Brace
174	‾			Overline
175	}			Closing Brace
176	|			Vertical Line
177	DEL		RUBOUT	Delete

Appendix I. FORTRAN Error Messages

This listing of error messages includes only those that are not self-explanatory.

COMPILER ERROR MESSAGES

1. EXPRESSION TYPE MISMATCHES "IF" TYPE

   Logical-IF and arithmetic-IF statement elements combined incorrectly -- examples:

   IF(A-1.)F=COS(T) <u>or</u> IF(B.AND.C)10,20,30

2. FILE NAMES CANNOT BE NUMERIC

   File references may not be unit numbers. File names may only be either filename variables or filename constants.

3. FILE REFERENCE IS NOT FILE NAME

   The variable or constant in question is not type filename.

4. FORMATS MUST BE EFN'S OR ARRAYS

   A format reference in an input/output statement must be either a statement number of array name.

5. ILLEGAL ADJUSTABLE DIMENSION VARIABLE

   The array or dimension to be adjusted is not a formal parameter to this subprogram.

6. ILLEGAL USAGE OPERATOR

   An operator other than + or - used illegally; e.g., A=*C or K=/I+J.

7. ILLEGAL USE OF $SN

   The value following the $ sign is not a valid statement number.

8. LIST LENGTHS DIFFER

   In a DATA statement, the number of constants given does not match the number of variables specified or implied.

9. MACHINE OR COMPILER LOGIC ERROR

   This type of error cannot be corrected by the user; contact your AE representative.

10. MUST HAVE 0 .LT. INTEGER .LT. 2**18

    The statement number indicated is out of the range of permissable values; i.e., is < 0 or $> 2^{18}$.

11. ODD COMMON CELL FOR TWO-WORD LENGTH VARIABLE

    The variable in question must start in an even cell; move it to an even location in the COMMON list (in source program).

12. OPTIONAL RETURN TOO BIG

    In RETURN i, i exceeds the number of nonstandard returns in the argument list.

13. PREVIOUSLY DIMENSION VARIABLE

    The dimension(s) of the variable in question were previously specified.

14. TOO MANY LIBRARY REQUESTS
    TOO MUCH BINARY
    TOO MUCH COMMON

    An internal compiler table has overflowed.

FORTRAN LOADER MESSAGES

145 TSS FORTRAN LOADER ERROR, CODE=xx

where 145 is the message number in the HELP subsystem and xx is as follows:

0-31	These codes (given in decimal) correspond to status codes listed under "Permanent File Activity", GE-625/635 GECOS-III TSS Programming Reference Manual, CPB-1514.
32	Your SAVE-file is not large enough to contain the whole object program.
33	Binary routine being saved is not in proper format. One or more object routines in the code being saved was probably created by some means other than TSS FORTRAN (e.g., GMAP). SYMREFS/SYMDEFS are not compatible for linking.
34	File being loaded contains too many subroutines (present limit is 64).
35	File being loaded is not in absolute format. This file was probably created by some means other than TSS FORTRAN.
36	Checksum error occurred in data blocks of file being loaded.
37	Checksum error occurred in control blocks of file being loaded.
38	Not used.
39	User's allotted file space has been exhausted.
40	Block count error while loading file. Data blocks are out of order or have been destroyed.
41	A file requested for loading cannot be found as described.

Index

DOCUMENT REVIEW SHEET

TITLE: GE-625/635 GECOS-III Time-Sharing FORTRAN

CPB #: 1566

From:

Name: ____________________

Position: ____________________

Address: ____________________

____

Comments concerning this publication are solicited for use in improving future editions. Please provide any recommended additions, deletions, corrections, or other information you deem necessary for improving this manual. The following space is provided for your comments.

COMMENTS: ____________________

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
Fold on two lines shown on reverse side, staple, and mail.

Please cut along this line

STAPLE STAPLE

FOLD

FIRST CLASS
PERMIT, No. 4332
PHOENIX, ARIZONA

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

GENERAL ELECTRIC COMPANY
PROCESSOR EQUIPMENT DEPARTMENT
13430 NORTH BLACK CANYON HIGHWAY
PHOENIX, ARIZONA 85029

ATTENTION: DOCUMENTATION B-107

FOLD

INFORMATION SYSTEMS

GENERAL ELECTRIC

LITHO U.S.A.