WRITER'S MASTER COPY

OBSOLETE

SDS

SCIENTIFIC DATA SYSTEMS

Reference Manual

DO NOT REMOVE

SDS 940 FORTRAN IV

FORTRAN IV COMMAND LANGUAGE

>|ENTER |	statement number |: statement number |	FORTRAN statement(s) (RET)
>COPY	statement number : statement number	/file name/ (RET)
>DELETE	|statement number| : statement number|| (RET)	
>LIST	|statement number| : statement number|| (RET)	
>RESEQUENCE	|old statement number range new statement number range| (RET)	
>REFERENCES	identifier |statement number : statement number| (RET)	
>DEFINITIONS	identifier |statement number : statement number| (RET)	
>EXECUTE		
>SAVE	/file name/ {SYMBOLIC / EDIT} (RET)	
>LOAD	/file name/ (RET)	

Price: $2.75

FORTRAN IV REFERENCE MANUAL

for

SDS 940 TIME-SHARING COMPUTER SYSTEM

90 11 15A

September 1967

SCIENTIFIC DATA SYSTEMS/1649 Seventeenth Street/Santa Monica, California

©1967, Scientific Data Systems, Inc.

Printed in U.S.A.

RELATED PUBLICATIONS

Title	Publication No.
SDS 940 Computer Reference Manual	90 06 40
SDS 940 Terminal User's Guide	90 11 18
SDS 940 Time-Sharing System Technical Manual	90 11 16
SDS 940 QED Reference Manual	90 11 12

NOTICE

The specifications of the software system described in this publication are subject to change without notice. The availability or performance of some features may depend on a specific configuration of equipment such as additional tape units or larger memory. Customers should consult their SDS sales representative for details.

CONTENTS

1. INTRODUCTION 1

Typographic Conventions 1
Operating Procedures 1
  Log-In 2
  Escape 2
  Exit and Continue 3
  Log-Out 3

2. FORTRAN IV PROGRAMS 3

Character Set 3
Statements 4
END Statement 4
Comments 4
FORTRAN Statement Labels 4
Statement Numbers 4

3. PROGRAM COMPILATION AND EXECUTION 5

General Description 5
FORTRAN IV Command Language 6
  ENTER Command 6
  COPY Command 8
  DELETE Command 8
  LIST Command 8
  RESEQUENCE Command 9
  DEFINITIONS Command 9
  REFERENCES Command 9
  EXECUTE Command 10
  SAVE Command 10
  LOAD Command 11
Compilation Diagnostics 12
Execution Diagnostics 12
Sample Program 12

4. DATA 16

Limits on Data Values 16
Constants 16
  Integer Constants 16
  Real Constants 16
  Double Precision Constants 17
  Complex Constants 17
  Logical Constants 18
  Hollerith Constants 18
Identifiers 18
Variables 18
  Scalar Variables 19
  Arrays and Array Variables 19
Functions 20

5. EXPRESSIONS 20

Arithmetic Expressions 20
  Evaluation Hierarchy 21
  Mixed Expressions 22
Relational Expressions 22
Logical Expressions 23
  Evaluation Hierarchy 24

6. ASSIGNMENT STATEMENTS 25

Replacement Statement 25
Label Assignment Statement 27

7. CONTROL STATEMENTS 27

GO TO Statements 27
  Unconditional GO TO Statement 27
  Assigned GO TO Statement 28
  Computed GO TO Statement 28
IF Statements 29
  Arithmetic IF Statement 29
  Logical IF Statement 29
DO Statement 29
CONTINUE Statement 30
PAUSE Statement 31
STOP Statement 31
Subprogram Control 31
  CALL Statement 31
  RETURN Statement 32
END Statement 32

8. INPUT/OUTPUT STATEMENTS 33

Input/Output Lists 33
  Simple List Items 33
  Do-Implied List Items 33
Free Format I/O 34
  ACCEPT Statement 34
  DISPLAY Statement 35
Formatted Input/Output 35
  OPEN Statement 36
  CLOSE Statement 36
  READ Statement 36
  WRITE Statement 37
  FORMAT Statement 37
Numeric Input Strings 49
Termination of Input Strings 49
Format and List Interfacing 50
FORMATs Stored in Arrays 51

9. DECLARATION STATEMENTS 52

Classification of Identifiers 52
  Implicit Declarations 52
  Explicit Declarations 53
Array Declarations 53
  Array Storage 54
  References to Array Elements 55
  DIMENSION Statement 55
  COMMON Statement 55
DATA Statement 56
Type Statement 57

10. SUBPROGRAMS 58

Function Subprograms 58
Library Functions 58
Statement Functions 62
FUNCTION Subprograms 62
SUBROUTINE Subprograms 63
Dummy Arguments 64

INDEX 69

APPENDIX

EXECUTION DIAGNOSTIC MESSAGES 66

TABLES

1. Evaluation of Logical Expressions 24
2. Mixed Variable Types and Expression Modes 26
3. Library Functions 59

1. INTRODUCTION

This manual is intended as a reference/operations manual for the SDS 940 Time-Sharing FORTRAN IV System and assumes the reader is familiar with the general principles of FORTRAN programming and with the 940 Executive System described in the SDS 940 Terminal User's Guide.

SDS 940 FORTRAN IV has been implemented as a group of programs operating as a subsystem to the time-sharing system. This implementation leads to a high degree of man-machine interaction since program compilation, modification, and execution are all controlled via a set of easily learned commands issued by the user at a teletype console.

In addition, FORTRAN IV permits incremental compilation of source program statements, which means that one or more program statements can be compiled independently. To modify a compiled statement, the user recompiles only that statement. To add text to a program, the user issues a command that directs the system to insert new material at a specified location. The results are very fast turnaround time for the programmer and minimal work for the system in keeping up with the program changes.

The present version of FORTRAN IV contains such extended features as mixed mode expressions, generalized subscripts, N-dimensional arrays, and identifiers of unlimited length. A flexible and easy to use input/output package is available to FORTRAN IV users. Implicitly formatted I/O offers easy data transfer to and from a character-oriented remote terminal. This feature can be of great value both to novices and to experienced programmers working with applications that do not require elaborate formatting. However, the full range of standard FORTRAN IV formatting features is also available.

Other capabilities of FORTRAN IV include free form entry of program statements and the ability to save and restore symbolic and object code.

TYPOGRAPHIC CONVENTIONS

For clarity, several conventions have been used throughout this manual. These are explained below:

1. Underscored copy in an example represents copy generated by the computer. Copy that is not underscored in an example must be typed by the user.

2. The following notations have been used to represent special keys on the teletype:

   (RET) represents the RETURN key.

   (ESC)[†] represents the ESCAPE key.

   (LF) represents the LINE FEED key.

3. Non-printing control characters are represented by an alphabetic character and a superscript c (e.g., D^c). The user depresses the alphabetic key and the Control (CTRL) key simultaneously to obtain a non-printing character.

OPERATING PROCEDURES

The standard procedure for gaining access to an SDS 940 time-sharing computer center from a teletype terminal is described in the SDS publication: SDS 940 Terminal User's Guide. The publication also includes information concerning the Executive System and the calling of various subsystems available to the terminal user. The following paragraphs summarize the standard procedures as they apply to FORTRAN IV users.

[†] In some 940 time-sharing systems the ALT MODE key is used instead of the ESCAPE key. Where (ESC) appears in this manual, ALT MODE may be substituted.

Log-In

To gain access to the computer, the following operating sequence is observed:

1. If the FD-HD (Full Duplex-Half Duplex) switch is present, turn the switch to the FD position. When the teletype is not connected to the computer (a condition sometimes called the Local Mode), this switch must be in the HD position.

2. Press the ORIG (originate) key, which is located at the lower right corner of the console directly under the dial. This key is depressed to obtain a dial tone before dialing the computer.

3. Dial the computer center number. When the computer accepts your call, the ringing will change to a high-pitched tone. There will then appear on the teletype a request that the user log in:

   PLEASE LOG IN:

4. The user must then type his account number, password, name and project code (if he has one) in the following format:

   PLEASE LOG IN: number password;name;project code (RET)

   Only persons who know the account number, password, and name, may log in under that particular combination. The following examples all illustrate acceptable practice.

   PLEASE LOG IN: A1PASS;JONES;REPUB (RET)

   PLEASE LOG IN: B4WORD;BROWN;DEMO (RET)

   PLEASE LOG IN: C6PW;SMITH; (RET)

   The optional 1-12 character project code is provided for installations that have several programmers using the same account number. The project code is not checked for validity.

   If the user does not correctly type his account number, password, and name within a minute and a half, a message is transmitted instructing him to call the computer center for assistance. The computer will then disconnect the user, and the dial and log-in procedure will have to be repeated.

5. If the account number, password (nonprinting), and name are accepted by the computer it will print READY, the date, and the time on one line.

   READY date, time

   -

   The dash indicates that the Executive is ready to accept a command.

6. In response to the dash the user types

   -FORTRAN

   >

   to call the FORTRAN IV compiler (the command may be abbreviated to the first three letters, FOR).

   FORTRAN IV will respond with a >, which means that it is awaiting a command.

Escape

The ESCAPE key can be used at any time to abort the current operation. If the FORTRAN compiler is in control at the time the key is depressed, control returns to the FORTRAN IV command language mode. Striking the ESCAPE key before terminating a FORTRAN IV command aborts the command. The system returns with >.

Exit and Continue

Depressing the (ESC) key several times in succession returns control to the Executive, which responds with a dash (-). If the user wants to return to FORTRAN IV without losing his program, and if he has not subsequently called another subsystem (e.g., BASIC, QED, CAL), he may type CONTINUE. The computer will type FORTRAN and return to it without any initialization. Meanwhile, nothing in core is destroyed.

Log-Out

When the user wishes to be disconnected from the computer, he depresses (ESC) several times in succession to return to the Executive and then types:

    -LOGOUT (RET)

or

    -EXIT (RET)

The computer will respond with the amount of hook-up (line) time charged to the user's account since the previous log-in procedure was completed.

2. FORTRAN IV PROGRAMS

An SDS 940 FORTRAN IV program is an ordered set of statements that describes a procedure to be followed by the computer and data to be processed by the program. Statements belong to one of two general classes:

- executable statements that perform computation, input/output operations, and program flow control.
- nonexecutable statements that provide information to the processor about storage assignments, data types, and program form and also provide information to the program during execution about input/output formats and data initialization.

Statements are usually entered on-line at a teletype console in a manner to be described in detail in the following chapter. The use and syntax of the various statements are explained in succeeding chapters.

Several conventions to be followed in writing programs differ from those used in card-oriented, batch processing systems and give to FORTRAN IV the flexibility necessary in a time-sharing system.

CHARACTER SET

The following characters may be used to form source statements:

Alphabetic:	A through Z
Numeric:	0 through 9
Special Characters:	+ - * ↑ / ( ) @ $ % & ? [ ] ' " . , : = < > and blank

The following control characters have special significance in FORTRAN IV:

(RET)	Terminates FORTRAN IV commands and FORTRAN IV statements.
(LF)	Allows statement to be continued to next line.
(ESC)	Aborts the current operation.
;	Terminates FORTRAN IV statements. May be used in place of (RET).

- \# Deletes the last character typed and may be used repetitively to delete more than one character. Note that # will not delete terminating semicolons.
- ← Deletes the entire statement currently being entered.

Hollerith input and Hollerith format fields may contain any printing teletype character except the control characters discussed above.

STATEMENTS

Statements may be entered at the teletype in a form-free format. It is unnecessary to follow the usual convention of beginning statements in column 7 of a line. Statements may begin anywhere on the line, including column one. A statement is terminated by either a (RET) or a semicolon. If a statement is terminated by a semicolon, the next statement may be typed immediately on the same line. A statement may then be continued from one line to the next by depressing (LF). Except for certain alphanumeric strings, blanks in a statement are ignored and may be used to aid readability.

END STATEMENT

A FORTRAN program must end with a statement consisting of the characters END. This statement indicates to the compiler that there are no more statements in the program; it has no effect upon execution. Since it is nonexecutable, the END statement should not be referenced by another statement.

COMMENTS

If the first nonblank characters of a statement are C: or *, the statement up to a semicolon or (RET) is treated as comments. Comments may appear anywhere in a program; they have no effect on execution.

Comments may be continued from line to line by depressing (LF).

FORTRAN STATEMENT LABELS

Any FORTRAN IV statement but COMMON, DIMENSION, and Type, may begin with a label consisting of any number of decimal digits. These numbers permit cross-reference between statements in the program. Leading zeros and blanks are ignored.

The following examples are equivalent:

```
  22
0022
 2 2
```

Labels are used for identification of addresses and must therefore be unique; i.e., no two statements may have the same number. No order of sequence is implied by the magnitudes of the statement labels. Nonreferenced statements need not be labeled.

FORTRAN statement labels should never be confused with statement numbers.

STATEMENT NUMBERS

When a source program is entered from the teletype or from a previously prepared file, FORTRAN IV assigns each statement a positive number in the range .001 through 999.999.

Once a program has been entered, these statement numbers may be used for text manipulation. For example, a user may modify or delete a statement by specifying its line number in the appropriate command.

A single number refers to one particular program statement. Two numbers separated by a colon indicate a range of statements. The range 22:30 specifies the statements from number 22 through 30, inclusive.

FORTRAN IV uses the following formula to calculate statement numbers:

$$\frac{R}{S} = I$$

where

R is the difference in the limits of the range.

S is the number of statements in the program minus 1.

I is the numerical difference (increment) between statement numbers.

If $R < S$, I is truncated after the first significant digit. If $R \geq S$, I is the integer portion of the result. For example, assume that the range is 1:4 and the number of statements is 5, then using the above formula

$$\frac{4-1}{5-1} = .75$$

The first statement would be statement number 1; the second statement would be statement number 1.7; etc.

3. PROGRAM COMPILATION AND EXECUTION

All communication from the programmer to the computer regarding text entry and compilation, program file manipulation, and program execution is conducted via the set of FORTRAN IV commands described in this chapter.

Programs are normally entered on-line at the teletype console. After logging in, the user calls FORTRAN IV with the Executive command

```
-FORTRAN
```

Upon receipt of this command, the Executive activates FORTRAN IV which prints a > to indicate readiness to receive commands.

GENERAL DESCRIPTION

The manner in which FORTRAN IV programs are entered into the compiler and compiled differs greatly from procedures used in batch processing environments. In the latter, the programmer typically prepares a complete program on a card or tape file, compiles this program file to obtain an object program, and then executes the object program. If modifications are necessary, the entire program must be recompiled.

In the SDS 940 FORTRAN IV environment, the user begins by issuing an ENTER command, followed by the source language statements (one or more) to be compiled. The FORTRAN statements may be entered one by one from the teletype keyboard or from a previously prepared file, using the COPY or LOAD command. The statements entered need not comprise a complete program. Once the user has entered his initial group of statements, he may add to the program, modify one or more statements, and delete code without recompiling the entire program. To add statements, he issues another ENTER command and uses statement numbers to indicate where the additional text is to be inserted. To modify existing statements, he merely recompiles the desired statements. The DELETE command performs total or partial deletion of program text.

The output from the compilation phase is a threaded list of "elements", each of which contains the encoded representation of a source language statement and certain directive information for use in structuring statements into the program.

Execution and program listing are controlled by appropriate FORTRAN IV commands. At the end of a session at the teletype the user may save his entire program, in either its symbolic or encoded forms, on a system file. This file may then be read in at the next session and the user may resume where he left off.

FORTRAN IV COMMAND LANGUAGE

All FORTRAN IV commands contain a one-word command identifier. Identifiers may be abbreviated to one, two, or three characters, depending on how many characters are required to distinguish a particular command from the others in the set. All characters typed by the user are checked for accuracy. The message

INVALID COMMAND IDENTIFIER

is printed if the system does not recognize the command.

Many commands may also include statement numbers that identify the program statements affected by the command.

The command identifier must be separated from successive fields by a space. If the user omits a required field, a message that describes the missing field is printed.

At any point during the entering of a command, the user may type in a question mark. A model of the command is then typed out. When a question mark is typed in as the first character of a statement, command models for all FORTRAN IV commands will be typed out.

All FORTRAN IV commands (except ENTER) must be terminated by a carriage return.

FORTRAN IV commands are grouped into eight functional categories:

	Function	Command
1.	Entering statements from the terminal	ENTER
2.	Entering statements from a previously prepared symbolic file	COPY
3.	Deleting statements from programs	DELETE
4.	Listing programs	LIST
5.	Resequencing programs	RESEQUENCE
6.	Locating variable and label definitions and references	DEFINITIONS and REFERENCES
7.	Executing programs	EXECUTE
8.	Maintaining and using program files	SAVE and LOAD

In the examples in this chapter, the abbreviation "sn" is used to specify a statement number.

ENTER Command

The ENTER command allows the user to enter original source statements or to modify previously prepared statements at the teletype.

Forms	Examples
>ENTER sn : sn FORTRAN statements D^c	>ENTER 1:4 ACCEPT [A]; B=SQRT [A]; DISPLAY [A, B]; (LF) END D^c
>ENTER sn FORTRAN statement {(RET) / D^c}	>ENTER 2 B = COS [A] {(RET) / D^c}

The ENTER command causes control to pass from the FORTRAN IV command mode to the FORTRAN IV compiler when the confirming D^c or (RET) is read.

The first form is used to create programs at the teletype. The compiler reads and compiles statements until a D^c is read. Statements may be separated by either (RET) or a semicolon and continued to another line by (LF). Line numbers are assigned according to the specified range.

The second form allows the user to enter a single FORTRAN statement into a program and is used typically to modify a previously entered statement or to insert an additional statement. If the number specified in the command has already been assigned to a statement in the program, the new statement replaces the old statement. If the number has not been assigned, it is inserted into the program according to numerical order.

Example

For the program

```
10.   DIMENSION X(10), Y(10);
20.   INTEGER Z;
30.   Z(I) = X(I) + Y (I);
40.   END;
```

the commands

```
>ENTER 25  DO  10  I=1, 10 (RET)
>ENTER 35  10  CONTINUE (RET)
>ENTER 20  INTEGER Z (10) (RET)
```

produce the program

```
10.   DIMENSION X(10), Y(10);
20.   INTEGER Z(10);
25.   DO  10  I=1,  10;
30.   Z(I) = X(I) + Y(I);
35.   10 CONTINUE;
40.   END;
```

Note that the statements to be inserted may be entered in any order; FORTRAN IV automatically inserts them into the program in numerical order.

The first form may also be used to modify or insert statements into an existing program. Statements associated with numbers within the specified range are replaced by the statements just entered. Statement numbers are evenly distributed within the range.

Example

```
>ENTER 10:40   DIMENSION X(10), Y(10) (RET)
INTEGER Z (RET)
Z(I) = X(I) (RET)
END                                  D^C
```

The command

```
>ENTER 20 : 30 (RET)
INTEGER Z(10) (RET)
DO  10  I = 1,  10 (RET)
Z(I) = X(I) + Y(I); 10 CONTINUE      D^C
```

produces the program

```
10.   DIMENSION X(10), Y(10);
20.   INTEGER Z(10);
23.   DO 10 I = 1,  10;
26.   Z(I) = X(I) + Y(I);
29.   10 CONTINUE;
40.   END;
```

If carriage returns are used to separate statements, semicolons will be inserted automatically and will appear on the program listing.

In both forms the word ENTER is optional.

COPY Command

The COPY command causes a saved symbolic file to be compiled as it is loaded into memory. The compiled statements are assigned statement numbers according to the specified range.

Form	Example
>COPY sn:sn /file name/ (RET)	>COPY 1:100 /FILE/ (RET)

The name of the file must be enclosed in slash marks.

Statements that contain errors are printed on the teletype and are then discarded. These statements may be corrected when the > character informs the user that compilation is completed.

Note that the FORTRAN IV COPY command differs from the Executive COPY command. The latter is used to create new files at the Executive level, whereas the FORTRAN IV COPY command loads into memory a previously created file.

DELETE Command

The DELETE command is used to delete the specified program statements.

Forms	Examples
>DELETE (RET) >DELETE sn (RET) >DELETE sn : sn (RET)	>DELETE (RET) >DELETE 3 (RET) >DELETE 40 : 60 (RET)

The first form deletes all statements in the current program. When this form is used, FORTRAN IV responds with

CLEAR PROGRAM?

The user must then type YES to have the entire program deleted or NO or (ESC) to abort the command.

The second and third forms delete only the specified statements.

Note that the DELETE command in FORTRAN IV differs from the Executive DELETE command. In the FORTRAN IV command mode, indicated by >, DELETE deletes the program or program statements that are currently in memory. At the Executive level, indicated by -, the DELETE command deletes a file from the user's file directory.

LIST Command

The LIST command lists the specified program statements on the teletype.

Forms	Examples
>LIST (RET) >LIST sn (RET) >LIST sn : sn (RET)	>LIST (RET) >LIST 5 (RET) >LIST 16 : 24 (RET)

The first form lists all statements in the current program, while the second and third forms list only the specified statements.

Statements are listed one per line. This command is typically used immediately after compilation to ascertain the statement numbers assigned to program statements.

RESEQUENCE Command

The RESEQUENCE command reassigns statement numbers to program statements.

Forms	Examples
>RESEQUENCE (RET) >RESEQUENCE old range new range (RET)	>RESEQUENCE (RET) >RESEQUENCE 30 : 60 1 : 30 (RET)

The first form reassigns statement numbers to all statements in the current program within the range 10:100. The second form reassigns new statement numbers to all statements within the old line number range.

DEFINITIONS Command

The DEFINITIONS command types out the statement, together with its statement number, that defines the specified variable or statement label.

Forms	Examples
>DEFINITIONS identifier (RET) >DEFINITIONS identifier sn : sn (RET)	>DEFINITIONS X (RET) >DEFINITIONS 7 30 : 50 (RET)

In the first form the definition of the specified identifier will be printed regardless of its position within the program. In the second form, the statement that defines the identifier will be printed only if it falls within the specified range.

Nondeclarative occurrences of the identifier are ignored.

Example

For the program

```
10.  DIMENSION X(10), Y(10);
20.  INTEGER Z(10);
30.  DO 10 I = 1, 10;
40.  Z(I) = X(I) + Y(I);
50.  10 CONTINUE;
60.  END;
```

The command

```
>DEFINITIONS  Z (RET)
```

types out

```
20.        INTEGER  Z(10);
```

and the command

```
DEFINITIONS  10 (RET)
```

types out

```
50.  10    CONTINUE;
```

REFERENCES Command

The REFERENCES command types out all statements, together with their statement numbers, that reference the specified variable, name, statement labels, or function references.

Forms	Examples
>REFERENCES identifier (RET) >REFERENCES identifier sn : sn (RET)	>REFERENCES I (RET) >REFERENCES 28 10 : 20 (RET)

In the first form all references to the specified identifier are printed on the teletype; in the second form references to the identifier within the specified range are printed.

Declarative occurrences of the identifier are ignored.

Example

For the program

```
10.  DIMENSION X(10), Y(10);
20.  INTEGER Z(10);
30.  DO 10 I = 1, 10;
40.  Z(I) = X(I) + Y (I);
50.  10 CONTINUE;
60.  END
```

The command

```
>REFERENCES Z (RET)
```

types out

```
40.      Z(I) = X(I) + Y(I);
```

and the command

```
>REFERENCES 10 (RET)
```

types out

```
30.      DO 10 I = 1, 10;
```

EXECUTE Command

The EXECUTE command causes control to transfer to the execution mode.

Form	Example
>EXECUTE (RET)	>EXECUTE (RET)

Execution begins with the first statement of the main program and terminates with the END statement. Control is then returned to the FORTRAN IV command mode.

SAVE Command

The SAVE command saves the specified program file in symbolic or edited form.

Forms	Examples
>SAVE /file name/ SYMBOLIC (RET) >SAVE /file name/ EDIT (RET)	>SAVE /FILE2/ SYMBOLIC (RET) >SAVE /FILE3/ EDIT (RET)

The name of the file must be enclosed in slash marks.

The SAVE command causes the user's file directory to be scanned. If the specified file name is found, FORTRAN IV responds with

OLD FILE

and replaces the program file in storage with the current version when the confirming (RET) is read. This feature is intended to protect the user from inadvertently writing on an old program file that he wants to preserve. The user may abort the command with (ESC) and assign a different name.

If the file name is not in the directory, FORTRAN IV responds with

NEW FILE

associates the specified name with the program that precedes the SAVE command, and saves it as a program file when the confirming (RET) is read. If the SAVE command is aborted while FORTRAN IV is writing the file to disc, unpredictable results will occur.

When SYMBOLIC is specified, only the source language is saved; no statement numbers or other information is included. A saved symbolic file is saved in a form acceptable to QED. Note that SAVE TELETYPE SYMBOLIC lists the program on the teletype.

If EDIT is specified, the compiler writes the compiled program, together with source language and statement numbers, onto the indicated file.

Since SAVE SYMBOLIC requires less disc space, it should be used in preference to SAVE EDIT, unless preservation of line numbers is essential. Recompilation time is generally negligible compared to disc reading time.

LOAD Command

The LOAD command loads a previously saved program file so that the user may resume working with it.

Form	Example
>LOAD /file name/ (RET)	>LOAD /FILE3/ (RET)

The name of the file must be enclosed in slash marks.

This command causes the compiler to scan the user's file directory. If the specified file name is found, it types either

SYMBOLIC FILE

or

EDIT FILE

indicating the file type. (RET) causes the file to be loaded in the indicated form, while (ESC) aborts the command. Statement numbers are assigned in the range 10 : 100 to a symbolic file. Statement numbers for an EDIT file are the same as those that were saved. The > character informs the user that the load process is completed.

If the file name is not in the user's directory, the system responds with

ERROR IN OPENING FILE

and returns control to the FORTRAN IV command mode. The user should then return to the Executive and enter the FILES command to ascertain the file names associated with his name and account number.

COMPILATION DIAGNOSTICS

Whenever a syntactical error is detected in a FORTRAN IV statement entered from the teletype, a warning bell rings and the statement in error is printed. An arrow (↑) is printed beneath the statement at the point beyond which compilation could not proceed. The user must take corrective action by either retyping the statement correctly or by skipping the statement and proceeding to the next one.

```
>ENTER 1 : 50 (RET)
A = 3.0 (RET)
X = Y*(Z-5 (RET)

X = Y*(Z-5;
          ↑
X = Y*(Z-5)  (RET)
.
.
.
END D^C
```

In this example an error occurred when the programmer omitted the right parenthesis. The computer immediately printed the statement and an arrow to indicate the point at which the statement failed to conform to an acceptable format. The user took corrective action by retyping the statement correctly. All statements with errors are discarded by the compiler, so that only the corrected version remains.

If the input to the compiler consists of a previously prepared file, the computer prints each statement in which there is an error and then discards it. It will not pause to await corrective action. When compilation is completed, the user may insert corrected statements, using the ENTER command. Alternatively, he may reread the file into QED to make his corrections.

The following messages are also printed by the compiler:

FUNCTION NOT IMPLEMENTED
PROGRAM REQUIRES END CARD
PROGRAM TOO LARGE

The compiler detects only syntactical errors; other types of errors are diagnosed during execution.

EXECUTION DIAGNOSTICS

If an error occurs during execution, the program is terminated and the statement that contains the error is printed at the teletype. See Appendix A for a list of these messages.

SAMPLE PROGRAM

The following program is designed to illustrate the use of the various FORTRAN IV commands and is not intended to be used as a guide in creating FORTRAN IV programs. Specifically, it is not necessary to use all the commands when composing FORTRAN IV programs.

PLEASE LOG IN: C8;W1B;DEMO (RET)	The user is requested to begin.
READY 6/30 10 : 21	
-FORTRAN (RET)	FORTRAN IV is called.
>ENTER 1:10 (RET)	FORTRAN IV responds with >.
INTEGER NEWER C (RET)	
OPEN(4, /SDS/, OUTPUT) (RET)	

```
DISPLAY ['PLEASE TYPE THE VALUES OF A, B AND C. "] (RET)
ACCEPT [A, B, C] (RET)
DISPLAY ['THANK YOU!!!!"] (RET)
NEW A = A**2 (RET)
NEW B = B**3 (RET)
NEW C = C**4 (RET)
DISPLAY ['NEW A = ", NEW A, ' NEW B = ", NEW B, ' NEW C = ", NEW C] (RET)
NEWER A = SQRT [NEW A] (RET)
NEWER B = NEW B**(1./3) (RET)
NEWER C = SQRT [SQRT [NEW C]] (RET)
WRITE(4, 10) NEWER A, NEWER B, NEWER C (RET)
10FORMAT(/$NEWER A = $E10.4/$NEWER B = $F10.4/$NEWER C = $I5) (RET)
PAUSE--RETURN TO THE EXEC & COPY FILE /SDS/ TO THE TELETYPE (RET)
```

Source program is entered.

```
END D^C
```

Multiple statement entry must be terminated by D^C.

```
>SAVE /FORT/ SYMBOLIC (RET)
```

Program is saved on disc in symbolic form and is assigned the name FORT.

```
NEW FILE (RET)
```

File assignment is confirmed.

```
>DELETE (RET)
CLEAR PROGRAM? YES (RET)
```

Program is deleted from user's memory.

```
>COPY 1 : 10 /FORT/ (RET)
```

FORT program file is compiled as it is loaded back into user's memory.

```
>LIST (RET)
1.   INTEGER NEWER C ;
1.6 OPEN(4, /SDS/, OUTPUT);
2.2 DISPLAY ['PLEASE TYPE THE VALUES OF A, B and C. "];
2.8 ACCEPT [A, B, C] ;
3.4 DISPLAY ['THANK YOU!!!!"] ;
4.   NEW A = A**2 ;
4.6 NEW B = B**3 ;
5.2 NEW C = C**4 ;
5.8 DISPLAY ['NEW A =", NEW A, ' NEW B = ", NEW B, ' NEW C = ", NEW C] ;
6.4 NEWER A = SQRT [NEW A] ;
7.   NEWER B = NEW B**(1./3) ;
7.6 NEWER C = SQRT [SQRT [NEW C]];
8.2 WRITE(4, 10) NEWER A, NEWER B, NEWER C ;
8.8 10FORMAT(/$NEWER A = $E10.4/$NEWER B = $F10.4 /$NEWER C = $I5) ;
9.4 PAUSE--RETURN TO THE EXEC & COPY FILE /SDS/ TO THE TELETYPE ;
10.  END ;
```

```
>RESEQUENCE 1:10 2:5 (RET)
```

Statement numbers are reassigned.

```
>LIST (RET)
2.    INTEGER NEWER C ;
2.2   OPEN(4,/SDS/,OUTPUT) ;
2.4   DISPLAY['PLEASE TYPE THE VALUES OF A, B AND C."];
2.6   ACCEPT [A,B,C] ;
2.8   DISPLAY ['THANK YOU!!!!"] ;
3.    NEW A = A**2 ;
3.2   NEW B = B**3 ;
3.4   NEW C = C**4 ;
3.6   DISPLAY ['NEW A =", NEW A, ' NEW B =", NEW B, ' NEW C =", NEW C];
3.8   NEWER A = SQRT [NEW A] ;
4.    NEWER B = NEW B**(1./3) ;
4.2   NEWER C = SQRT [SQRT [NEW C]];
4.4   WRITE(4,10)NEWER A, NEWER B, NEWER C ;
4.6   10FORMAT(/$NEWER A = $E10.4/$NEWER B = $F10.4 /$NEWER C = $I5) ;
4.8   PAUSE--RETURN TO THE EXEC &COPY FILE/SDS/TO THE TELETYPE ;
5.    END ;
>REFERENCES NEWERC (RET)
4.2   NEWER C = SQRT [SQRT [NEW C]];
4.4   WRITE(4,10) NEWER A, NEWER B, NEWER C ;
```

FORTRAN IV prints statements that contain specified references.

```
>DEFINITIONS 10 (RET)
4.6   10FORMAT(/$NEWER A = $E10.4/$NEWER B = $F10.4/$NEWER C = $I5) ;
```

FORTRAN IV prints statement that defines label 10.

```
>EXECUTE (RET)
2.2   OPEN(4,/SDS/,OUTPUT);
FILE NAME NOT IN DIRECTORY
```

Program cannot be executed because output file is not in the user's file directory.

```
> (ESC)
> (ESC)
> (ESC)
```

Control is returned to the Executive.

```
-COPY TELETYPE TO /SDS/ (RET)
NEW FILE (RET)
THIS IS A DUMMY FILE    D^c
-CONTINUE (RET)
```

Control is returned to FORTRAN IV.

```
FORTRAN
>EXECUTE
PLEASE TYPE THE VALUES OF A, B AND C.
2,4,6 (RET)
THANK YOU!!!!
```

```
NEW A = 4  NEW B = 64  NEW C = 1296
PAUSE--RETURN TO THE EXEC & COPY FILE/SDS/TO THE TELETYPE
>(ESC)
>(ESC)
>(ESC)
-COPY/SDS/TO TELETYPE (RET)
NEWER A = .2000E + 01
NEWER B = 4.0000
NEWER C = 6
-CON (RET)

FORTRAN
>LIST (RET)
 2.     INTEGER NEWER C ;
 2.2    OPEN(4,/SDS/,OUTPUT) ;
 2.4    DISPLAY ['PLEASE TYPE THE VALUES OF A, B AND C."] ;
 2.6    ACCEPT [A,B,C] ;
 2.8    DISPLAY ['THANK YOU!!!!"] ;
 3.     NEW A = A**2
(ESC)
>DELETE 2.4 (RET)
>LIST (RET)
 2.     INTEGER NEWER C ;
 2.2    OPEN(4,/SDS/,OUTPUT) ;
 2.6    ACCEPT [A,B,C] ;
 2.8    DISPLAY ['THANK YOU!!!!"];
 3.     NEW A = A**2 ;
 3.2
> (ESC)
>(ESC)
>(ESC)
-LOGOUT (RET)
TIME USED 0 : 10 : 23
```

Contents of output file/SDS/is printed.

The Executive command CONTINUE may be abbreviated to the first three letters.

List process is aborted.

Statement 2.4 is deleted.

List process is aborted and control is returned to the Executive.

User requests to logout.

Ten minutes and 23 seconds have elapsed since user logged in.

4. DATA

Numerical quantities – constants and variables – in FORTRAN IV are a means of identifying the nature of the numerical values encountered in a program. A constant is a quantity whose value is explicitly stated. For example, the integer 5 is represented as "5"; the number π, to three decimal places, as "3.142". A variable is a numerical quantity that is referenced by a symbolic name rather than by its explicit appearance in a program statement. During execution of the program, a variable may take on many values rather than being restricted to one.

All data processed by a FORTRAN IV program can be classed into six groups: integer, real, double precision, complex, logical, and Hollerith.

LIMITS ON DATA VALUES

Both integer and real (or "floating point") data can be assigned any value in the approximate range 10^{-77} to 10^{76}. Both kinds of data are stored in floating point form, using two words or 48 bits: a 38-bit mantissa, 9-bit exponent, and a sign bit. Both integer and real data have an associated precision of 11+ significant digits. That is, numbers with 11 significant digits will be accurate, while numbers with 12 significant digits will be accurate for values up to $2^{38}-1$. Numbers greater than this will lose accuracy in the least significant position.

Double precision data may approximate the identical set of values as single precision floating point data, but have an associated precision of 18+ significant digits.

Complex data are approximations of complex numbers, taking the form of an ordered pair of real data. The first of the two real data approximates the real part, and the second real datum approximates the imaginary part of the complex number. The values each part may be assigned are identical to the set of values for real data.

Logical data can acquire only the values "true" or "false".

Hollerith data represent character string values. The set of values that each character in the string may assume are given in Chapter 2, in the discussion on the FORTRAN character set. A Hollerith datum is stored in two computer words in ASCII[†] code. Characters are stored left-justified with trailing blanks.

CONSTANTS

Constants are data that do not vary in value and are referenced by naming their values. Constants may be any type of data. For constants with positive values the plus character (+) need not be present.

Integer Constants

Integer constants are represented by strings of decimal digits optionally preceded by a sign character.

Form	Examples	
±n	392 -13	+997263 1234567

where n is a string of digits, and the plus sign is optional.

Real Constants

Real constants are represented by strings of digits with a decimal point and/or an exponent. The exponent follows the numeric value and consists of the letter E followed by a signed or unsigned integer that represents the power of ten by which the numeric value is to be multiplied. Thus, the following forms are permissible:

[†] American Standard Code for Information Interchange.

Forms				Examples			
±n.m	±n.	±.m		-394.6238763	5.	.39653	
±n.mE±e	±n.E±e	±.mE±e	±nE±e	-3946.238763E-5	1.E1	.5E-2	-1E-1

where n, m, and e are strings of digits, and the plus sign preceding e is optional.

The following forms are all equivalent:

+0.567E+05	.567E5	5.67E+4	56700.0
567000E-01	.567E05	56700.E0	56700E-00

Since any real constant may be represented in a variety of ways, the user can choose the form most convenient for his purpose.

Double Precision Constants

Double precision constants are formed exactly like real constants, except that the letter D is used as the exponent instead of E. To denote a constant specifically as double precision, the exponent must be present. Thus, a double precision constant may be written in any of the following four forms:

Forms				Examples			
n.mD±e	n.D±e	.mD±e	nD±e	6.88D22	763.D1	-.098734D+5	763D1

where n, m, and e are strings of digits, the plus sign preceding e is optional, and D signifies a double precision constant.

Generally, it is unnecessary to form a double precision constant, even when precision greater than 11+ digits is desired. Any constant that appears in a double precision expression will become a double precision constant (with 18+ digits of accuracy) even if it is written as a real constant. The only effect a double precision constant can have is to cause an expression which would otherwise be real or integer to be computed in double precision mode, as in the case:

```
X = A*B/Y + 0.D0
```

The value of a double precision constant may not exceed the limits for double precision data. Double precision constants specified with more significance than precision allows are truncated to the 18+ most significant digits.

Complex Constants

Complex constants are expressed as an ordered pair of constants in the format:

Form	Example
(c_1, c_2)	(98.7, 25.05)

where c_1 and c_2 may be integer or real constants. The parentheses and comma characters are required. Integer constants are converted to real constant approximations of their values. The complex constant (c_1, c_2) is interpreted as meaning $c_1 + c_2 i$. The following complex constants have values as indicated:

(1.34,52.01	=	1.34+52.01i
(98344,.34452E + 02)	=	98344.0+34.452i
(-1.,-1000)	=	-1.0 - 1000.0i
(2.3,0)	=	2.3+0i
(0,4.5)	=	0+4.5i
(2.7E1,0.8)	=	27.0+0.8i

Neither part of a complex constant may exceed the value limits established for real data.

Logical Constants

Logical constants may assume either of two forms:

. TRUE.   . FALSE.

where these forms have the values "true" and "false" respectively.

Hollerith Constants

Hollerith constants are represented in the form

Form	Examples			
nHs	4HFOUR	3HYOU	2H$$	1H+
	6HOH BOY	3HOH?	2HX=	1HH

where n is an unsigned integer constant of the set (0, 1, 2, 3, 4, 5, 6) and s is a string of characters whose length exactly corresponds to the value of n. The character H appears in that form. Each character in a Hollerith constant may be one of the set of characters discussed in Chapter 2.

Hollerith constants may be assigned to real variables only. Since Hollerith constants are stored 3 characters per 24-bit word, and real data use 2 words each (48 bits), a maximum of 6 characters is allowed in the Hollerith constant. If less than 6 are used, the characters are stored left-justified with trailing blanks.

If a variable to which a Hollerith constant is assigned is to be output, an A format specification should be used (see Chapter 8).

IDENTIFIERS

Identifiers are strings of letters and decimal digits, the first of which must be a letter. Identifiers are used to name variables as well as subprograms and subprogram arguments. Identifiers in FORTRAN IV may be of any length. Embedded blanks are ignored.

There are no restricted identifiers, but for clarity, it is not advisable to use identifiers which correspond to SDS 940 FORTRAN statement types.

Examples:

X   A345Q   J3   QUANTITY   FIRST ONE

ELEVATION   I   L987564   DIFFERENTIAL

VARIABLES

Variables are data whose values may vary during program execution and which are referenced with an identifier. Variables may be any of the data types.

If a variable has not been explicitly assigned to a particular data type (Chapter 9), the following conventions are assumed:

- Variables whose identifiers begin with the letters I, J, K, L, M, N are integer data.
- Variables whose identifiers begin with any other letter are real data.

Consequently, double precision, complex, and logical variables must be explicitly declared as such. The values assigned to variables may not exceed the limits established for the applicable data types.

Scalar Variables

A scalar variable is a single datum entity and is accessed via an identifier of the appropriate type.

Examples:

```
I1
EXPONENT
NAME
XXX8
```

Arrays and Array Variables

An array is an ordered set of data that may be referenced and altered in a program. The set as a whole is named by an array identifier according to the rules discussed above for variables. The elements of the array, called array variables, are referenced by the array identifier followed by an expression, called a subscript, which describes the element's position within the array.

For example, A(4) refers to the fourth element in a set of elements called A. This would be a one-dimensional array, or vector. A two-dimensional array is considered arranged into columns and rows. An element in a two-dimensional array is referenced as A(I, J) where I refers to a row element and J refers to a column element. For example, in the set of numbers

```
111   222   333
444   555   666
777   888   999
```

if the entire set is called B, then the element 666 is referenced as B(2, 3). B is called a "3 by 3" array or matrix.

Subscripts. Subscripts may assume the following form:

Form	Example
$(s_1,s_2,s_3,\ldots,s_n)$	(2,6.5,5.3)

where the s_i are any expressions of integer or real mode, and n is the value of the number of dimensions associated with the array. The parentheses and comma characters are required.[†] Real expressions used as subscripts are truncated to integer values.

Examples:

Array Name	Array Variable
MATRIX	MATRIX (3,9)
CUBE	CUBE (J*4,P,3.6)
A	A (Q/I+U-M)
J	J(7.5E+2)

Nested subscripting is permissible; that is, subscripts themselves may be subscripted. There is no limit on the level of nesting.

Examples:

```
ALPHA(I(J))
MATRIX (I(J(K)))
```

[†] In all examples given in this format, unless otherwise stated, multiple period characters signify possible additional specifications and are not actually present in the FORTRAN code.

FUNCTIONS

Functions are subprograms that are referenced as basic elements in expressions. A function acts upon one or more quantities, called its arguments, and produces a single quantity, called the function value. The appearance of a function reference constitutes a reference to the value produced by the function, when operating on the given arguments.

A function reference is denoted by the identifier that names the function, followed by a list of arguments enclosed in brackets.

Form	Examples
$f[a_1, a_2, \ldots, a_n]$	SIN [A+B] KOST [ITEMNUM]

where

f is the name of the function, and

a_i are arguments. Arguments may be constants, scalar variable references, array element references, array names (no subscripts), expressions, or subprogram identifiers.

Functions are classified in the same way as variables; that is, unless the type is specifically declared, the IJKLMN rule applies. The type of a function is not affected by the type of its arguments.

5. EXPRESSIONS

Expressions are strings of operands separated by operators. Operands may be constants, variables, or function references. An expression may contain subexpressions; i.e., expressions enclosed in parentheses. Operators may be unary, operating on a single operand, or they may be binary, operating on pairs of operands.

Expressions may be classified as arithmetic, relational, or logical. All expressions yield a single, unique value when evaluated.

ARITHMETIC EXPRESSIONS

An arithmetic expression is a sequence of constants, variables, or function references connected by arithmetic operators.

The arithmetic operators and their associated connotations are as follows:

Operator	Operation
+	Addition (binary) or Positive (unary)
-	Subtraction (binary) or Negative (unary)
*	Multiplication
/	Division
** or ↑	Exponentiation

Expressions may consist of a single basic element; i.e., a constant, variable, or function. For example:

```
3.1415
X(N)
SQRT[ALPHA]
```

Basic elements may be combined through use of the arithmetic operators to form compound expressions. For example,

```
A + B
PI*RADIUS**2
SQRT THETA*[THETA]
```

Compound expressions may be enclosed in parentheses to form subexpressions. For example,

```
(A + B)/(C + E)
-((M - N)*(Z - Q(J)))
```

Evaluation Hierarchy

The expression A+B/C could be evaluated as

```
(A+B)/C
```

or as

```
A+(B/C)
```

To avoid the possibility of such ambiguities, various rules governing precedence of evaluation have been formulated. The evaluation hierarchy is as follows:

1. The innermost subexpression, followed by the next innermost subexpression, until all expressions have been evaluated.

2. The arithmetic operations in the following order of precedence:

Operation	Operator	Order
Exponentiation	** or ↑	1 (highest)
Multiplication and Division	* /	2
Addition and Subtraction	+ -	3

Several additional conventions are necessary:

1. At any one level of evaluation, operations of the same order of precedence are evaluated from left to right. Consequently, I/J/K/L is equivalent to ((I/J)/K)/L.

2. As in algebraic notation, parentheses are used to define evaluation sequences explicitly. Thus, $\frac{A+B}{C}$ is written as (A+B)/C.

3. The sequence "operator operator" is permissible if the expression can be evaluated when the second operator is interpreted as unary. Thus A*-B is interpreted as A*(-B).

As an illustration of the above rules of precedence, the expression

```
A*(B+C*(D-E/(F+G)-H)+P(3))
```

is evaluated in the following equivalent sequence:

$r_1 = F + G$

$r_2 = E/r_1$

$r_3 = D - r_2 - H$

$r_4 = C*r_3$

$r_5 = B + r_4 + P(3)$

$r_6 = A*r_5$

where the r_i are the various levels of evaluation.

Mixed Expressions

Arithmetic expressions may contain references to data or functions of the integer, real, double precision, or complex types. References to data, subexpressions, or functions of the logical type are excluded from arithmetic expressions, except when they appear as function arguments. When arithmetic expressions contain references of more than one type, they are called mixed expressions.

Mixed expressions are evaluated in the mode of the highest order of reference:

Type	Precedence
Complex	1 (highest)
Double precision	2
Real	3
Integer	4

The following rules also govern evaluation of mixed expressions:

1. Expressions appearing as subscripts or function arguments are evaluated separately in their own modes and have no effect on the mode of the expression in which they are contained.
2. Exponents may be integer or real.
3. Double precision values are truncated to real value precision when they appear in complex mode expressions.
4. Values of expressions, subexpressions, and terms are restricted to those limits associated with the mode of the expression.
5. Values of double precision, real, or integer mode that appear in complex mode expressions are assumed to have imaginary parts of zero value.

RELATIONAL EXPRESSIONS

A relational expression consists of arithmetic expressions of integer or real mode, separated by relational operators that cause the expressions to be compared. Evaluation results in one of the two logical values "true" or "false".

In general, the form of a relational expression may be written

Form	Example
$e_1 r_1 e_2$. AND . $e_2 r_2 e_3$. . . . AND . $e_{n-1} r_{n-1} e_n$	A . LT . B . AND . B . GT . C

where the e_i are arithmetic expressions and the r_i are relational operators.

The following table shows the relational operators and their meanings:

Operator	Meaning
.LT.	Less than
.LE.	Less than or equal to
.EQ.	Equal to
.NE.	Not equal to
.GT.	Greater than
.GE.	Greater than or equal to

A relational expression has the value "true" only if all comparisons in the expression are true. For example,

1 .LT. 6	is true.
0 .GT. 8	is false.
7 .GT. 2 .AND. 2 EQ. 5	is false, because 2 .EQ. 5 is false.
0 .LT. (2. **N)	is always true, but
0 .LT. -(2.**N)	is always false, while
X .GT. 5 .AND. 5 NE. I	will be true or false depending upon the values of X and I.

If one expression is integer and the other real, the two expressions are first evaluated, each in its own mode; then the value of the integer expression is converted to real mode and a real comparison made.

It is not permissible to nest relational expressions as in the case

(L .GE. (X .GT. 0.2345E6))

where X .GT. 0.2345E6) is a relational subexpression rather than an arithmetic expression, as the definition of relational expressions requires. Also,

N .GE. Y * (T *((O + P) .NE. C) - Q) .LT. Z

is meaningless since ((O + P) .NE. C) is a relational subexpression which is not allowable in the otherwise arithmetic expression following the .GE. operator.

LOGICAL EXPRESSIONS

A logical expression is an expression of the form:

Form	Example
$e_1c_1e_2c_2e_3c_3 \ldots e_n$	C .OR. B .AND. X

where the e_i are logical elements, and the c_i are the binary logical operators.

Evaluations of logical expressions result in one of the two values "true" or "false".

Logical elements are defined as one of the following entities:

1. A logical variable or function reference

2. A logical constant

3. A relational expression
4. Any of the above enclosed in parentheses
5. A logical expression enclosed in parentheses
6. Any of the above preceded by the unary logical operator .NOT.

There are four logical operators:

Operator	Type
.NOT.	unary
.AND.	binary
.OR.	binary
.EOR. (exclusive OR)	binary

Logical expressions are evaluated as follows (the letter "e" denoting a logical element):

1. .NOT. e — true only when e is false.
2. e_1 .AND. e_2 — true only when both e_1 and e_2 are true.
3. e_1 .OR. e_2 — true when either or both e_1 and e_2 are true.
4. e_1 .EOR. e_2 — true when either but not both e_1 and e_2 are true.

These rules are illustrated in the following table:

Table 1. Evaluation of Logical Expressions

Expression Values		Logical Operator			
		.NOT. e	.AND.	.OR.	.EOR.
e True	—	False	—	—	—
e False	—	True	—	—	—
e_1 False	e_2 False	—	False	False	False
e_1 True	e_2 False	—	False	True	True
e_1 False	e_2 True	—	False	True	True
e_1 True	e_2 True	—	True	True	False

Evaluation Hierarchy

In a manner similar to that discussed for arithmetic expressions, parentheses are used to define explicit evaluation sequences. Consequently,

A .AND. B .OR. Q(3) .NE. X

does not have the same meaning as

A .AND. (B .OR. Q(3) .NE. X)

where (B .OR. Q(3) .NE. X) may be called a logical subexpression.

The evaluation hierarchy of logical expressions is:

1. Arithmetic expressions

2. Relational expressions (the relational operators are all of equal precedence)

3. The innermost logical subexpression, followed by the next innermost logical subexpression, etc.

4. The logical operations in the following precedence:

Operator	Order
.NOT.	1 (highest)
.AND.	2
.OR.	3
.EOR.	4

Note: It is permissible to have two continguous logical operators only when the second operator is .NOT.. In other words,

e_1 .AND. .OR. e_2

is illegal, while

e_1 .AND. .NOT. e_2

is legal.

6. ASSIGNMENT STATEMENTS

The SDS 940 FORTRAN IV language is comprised of five types of statements:

- Assignment Statements
- Control Statements
- Input/Output Statements
- Declaration Statements
- Subprogram Statements

Each type of statement performs a specific function. Assignment statements are discussed in this chapter; subsequent chapters are devoted to discussion of the other statements.

REPLACEMENT STATEMENT

The Replacement statement specifies 1) an expression to be evaluated and 2) a variable, called the statement variable, to which the expression value is to be assigned.

Form	Examples
v = e	A = B Q(I) = Z**2 + N* (L-J) L = B .OR. .NOT. C .AND. R. NE.23.93

where v is a variable name and e is an expression. Note that the sign (=) denotes replacement rather than equality. Thus Y=Y+1 is a valid statement meaning "add one to the value of Y and assign the resulting value to Y".

When the mode of the expression e is not the same as the variable type for v, the variable is assigned values as indicated in the following table.

Table 2. Mixed Variable Types and Expression Modes

Rules for Assignments of e to v		
If v type is	**and e type is**	**assignment rule is**
Integer	Integer	Assign
	Real	Fix and Assign
	Double Precision	Fix and Assign
	Complex	Illegal
	Logical	Illegal
Real	Integer	Float and Assign
	Real	Assign
	Double Precision	DP Evaluate and Real Assign
	Complex	Illegal
	Logical	Illegal
Double Precision	Integer	DP Float and Assign
	Real	DP Evaluate and Assign
	Double Precision	Assign
	Complex	Illegal
	Logical	Illegal
Complex	Integer	Illegal
	Real	Illegal
	Double Precision	Illegal
	Complex	Assign
	Logical	Illegal
Logical	Integer	Illegal
	Real	Illegal
	Double Precision	Illegal
	Complex	Illegal
	Logical	Assign

Notes:

1. Assign means transmit the resulting value, without change, to the variable.
2. Fix means truncate any fractional part of the result and transform that value to the form of an integer datum.
3. Float means transform the value to the form of a real datum.
4. DP Evaluate means evaluate the expression double precision according to the usual rules of evaluation.
5. Real Assign means transmit as much precision of the most significant part of the resulting value as a real datum can contain.
6. DP Float means transform the value to the form of a double precision datum, retaining in the process as much of the precision of the value as a double precision datum can contain.

LABEL ASSIGNMENT STATEMENT

Label Assignment statements are used to assign to a variable the location of a statement.

Form	Examples
ASSIGN k TO v	ASSIGN 153 TO LABEL ASSIGN 603 TO FLAG1

where k is a statement label and v is a scalar variable reference of any data type.

Once a statement label has been assigned to a variable, the variable must not be referenced except as a statement label. Thus in the sequence

```
ASSIGN 101 TO A
C = A/B
```

will cause an execution time diagnostic because the value of A is undefined.

Note that the statement M = 5 cannot be substituted for ASSIGN 5 to M and vice versa because the integer "5" is implied in the first case, and the label "5" in the second.

The use of such assignments is discussed in the next chapter, in the section on Assigned GO TO Statements.

7. CONTROL STATEMENTS

Each statement in a FORTRAN IV program is processed in the order of its appearance in the source program unless this sequence is interrupted or modified by a control statement. If program control is to be transferred to a particular statement, that statement must be identified by a label (see Chapter 2).

In general control statements may be used to:

- Provide unconditional transfer of control to other statements in the program
- Test variables and provide conditional transfer of control to other statements in the program
- Execute a particular sequence of statements repeatedly a specified number of times
- Provide branching to and return from subprograms

The program control statements available in SDS 940 FORTRAN IV include GO TO, IF, DO, CONTINUE, PAUSE, STOP, CALL, RETURN, and END.

GO TO STATEMENTS

There are three forms of GO TO statements: unconditional, assigned, and computed.

Unconditional GO TO Statement

The Unconditional GO TO statement provides a means to unconditionally transfer control to another statement in the program.

Form	Examples
GO TO k	GO TO 5 GO TO 800

where k is the statement label of an executable statement.

The result of execution of this statement is that the next statement executed is the statement whose label is k.

Assigned GO TO Statement

The Assigned GO TO statement transfers control to a statement referenced by a variable label defined previously in an ASSIGN statement (see Chapter 6).

Forms	Examples
GO TO v	GO TO G
GO TO v, $(k_1, k_2, k_3, \ldots, k_n)$	GO TO G, (117,56,101)

where v is a variable appearing in a previously executed ASSIGN statement and the k_i are statement label references.

Control is transferred to the statement whose location has been assigned to the variable v.

If the second form is used, each label in the list must be defined in the program or subprogram segment in which the GO TO statement appears (i.e., must be the label of a program statement). This form serves no purpose other than to provide compatibility with other processors. The comma and parentheses characters must appear as shown.

For example, the statements

```
ASSIGN 5371 to G
GO TO G
```

will cause transfer of control to the statement labeled 5371. The optional form would be

```
ASSIGN 5371 TO GO
GO TO G, (117, 56, 101, 5371)
```

Computed GO TO Statement

The Computed GO TO statement allows transfer of control to one of a group of statements, the particular statement chosen depending on conditions at run time.

Form	Examples
GO TO $(k_1, k_2, k_3, \ldots, k_n)$, e	GO TO (98, 65, 405, 3), R GO TO (5, 6, 7), T**2 - 1

where the k_i are statement labels and e is an expression of integer or real mode.

The comma character preceding e is optional.

Control is transferred to the statement whose label is k_j. where j is the integer value of the expression e. The value of the expression must be greater than zero and less than or equal to n; that is, $0 > j \leq n$. Real mode expressions are evaluated and than truncated to integer value.

In the first example above, if the expression R has the value 3, then control will be transferred to the statement labeled 405. If the expression (T**2 - 1) in the second statement has the value 1.56 control will be transferred to the statement labeled 5.

IF STATEMENTS

IF statements are conditional transfer statements that allow the programmer to change the logical flow of a program on the basis of a test. There are two types of IF statements: arithmetic and logical.

Arithmetic IF Statement

The form for the Arithmetic IF statement is

Form	Examples

IF (e) k_1, k_2, k_3	IF(G + B(I)) 76, 4, 3
	IF(X - Y) 100, 250, 3000
	IF(I) 1, 2, 3

where e is an expression of integer or real mode and k_1, k_2, and k_3 are statement labels. If the value of e is less than 0, transfer is to k_1; if the value of e is equal to 0, transfer is to k_2; if the value of e is greater than 0, transfer if to k_3.

A comma character may optionally precede k_1.

Examples

Statement	Expression Value	Transfer To
IF(I)1, 2, 3	47802	3
IF(C(J, 10)/4), 23, 12, 8	-.098433	23
IF(A+B(I))44, 33, 22	0.0	33

Logical IF Statement

The Logical IF statement is represented as

Form	Examples
IF (e) s	IF(E .OR. D) GO TO 3135 IF (A .AND. G), IF(C. NE. K), ON = .TRUE.

where e is a logical mode expression and s is any executable statement.

A comma may optionally precede the statement s.

The statement s is executed if the expression e has the value "true"; otherwise, the next executable statement following the Logical IF statement is executed. The statement following the Logical IF will be executed in any case after the statement s, unless the statement s causes a transfer to occur, as in the first example above.

Note that the entire construct IF (e) s is treated as a single statement, which allows a Logical IF to control another Logical IF. This is illustrated in the second example.

DO STATEMENT

The DO Statement is used to control repetitive execution of a group of statements.

Form	Examples
DO k v = e_1, e_2, e_3	DO 10 I = 1, 10 DO 12 J = 2, 98, 2 DO 15 V = END, START, -.05

where k is a statement label, v is a reference to a scalar variable of integer or real mode, and e_1, e_2, and e_3 are expressions of integer or real mode.

An optional comma character may be placed between k and v.

The DO statement causes repeated execution of all statements within its range. The range of a DO extends from the first executable statement following the DO statement up to and including statement k.

The scalar variable v is called the index of the DO statement. It is used to identify the repetition currently being performed. The value of e_1 represents the initial value of the index; the value of e_2 represents the limiting or terminal value of the index; and e_3 the incrementing quantity. If e_3 is omitted, it is assumed to be 1.

The initial execution is always performed, regardless of the values of the limit and increment. After each execution of the range, the increment value is added to the value of the index, and the result is compared with the limit value. If the value of the index is not greater than the limit, the range is executed again, using the new value of the index. (In case the increment value is negative, another execution will be performed if the new value of the index is not less than the limit value.)

When the index value exceeds (or if decrementing is less than) the limit value, control passes to the statement immediately following statement k. Exit may also be effected by a transfer from within the range of the DO statement.

Consider this example:

```
DO 999, I = 1, 5, 2,
```

MEANING: Execute all statements immediately following, up to and including statement number 999, first for I = 1, next for I = 3, and last for I = 5. Then transfer control to the statement following statement number 999. Thus, the loop will be executed a total of 3 times.

The terminal statement of a DO range (k) may be any executable statement. However, the programmer should exercise care if the terminal statement is a transfer (GO TO, IF); the consequences can be determined by inspection. Incrementing and testing will not take place if k is a transfer statement.

If a transfer is made out of the range of a DO before all iterations have been completed, the value of v will be that during which the transfer occurred.

The value of the variable v may be modified by any form of assignment statement within the range of the DO, and may also be modified by a subprogram called within the range of the DO.

A transfer into the range of a DO may only occur if there has been a prior transfer out of the range. In fact, the statements executed "outside" the range will then be considered part of the DO range.

A DO-loop may include other DO-loops, provided that the range of each "inside" or "nested" DO statement is contained completely within the range of an "outside" DO statement. In other words, the ranges of two DO statements may not partially overlap one another. There is no limit to the level of nesting. The same statement may be used as the terminal statement for any number of DO statements.

If the programmer wishes to avoid terminating a loop with a transfer statement, he may use the CONTINUE statement as a dummy end for the loop.

CONTINUE STATEMENT

The CONTINUE statement is a dummy statement used primarily to serve as a target point for transfers, particularly as the last statement in a DO loop. At the end of the range of a DO, the CONTINUE statement in effect means "do nothing but proceed to modify and test the index".

For example, in the sequence

```
DO 5  I = 1,  MAX
.
.
.
GO TO 5
.
.
.
X = SUM
.
.
.
5  CONTINUE
```

if the GO TO is intended to begin another execution of the DO loop, without performing the statement X = SUM, the CONTINUE statement provides the necessary target address.

PAUSE STATEMENT

The PAUSE statement temporarily halts execution of a program.

Forms	Examples
PAUSE	PAUSE
PAUSE c	PAUSE 777

where c is any string of characters.

The word PAUSE will be displayed at the teletype, as will the string c if it is specified. By typing any character, the programmer can cause execution to continue with the statement immediately following PAUSE.

STOP STATEMENT

The STOP statement terminates the program and returns control to the FORTRAN IV command mode.

Forms	Examples
STOP	STOP
STOP c	STOP 777

The character c has the same meaning as for tne PAUSE statement.

SUBPROGRAM CONTROL

The two statements discussed below provide transfer of control between subprograms and calling programs (see Chapter 10 for a general description of subprograms).

CALL Statement

The CALL statement causes a transfer of control to a SUBROUTINE subprogram.

Forms	Examples
CALL p	CALL DUMP
CALL p $[a_1,a_2,\ldots,a_n]$	CALL FACTOR [A + 1,2*A]

where p is the identifier of the subroutine, and the a_i are arguments required by the subroutine.

If the called subroutine does not require an argument list, the first form above is used.

Arguments appearing in a CALL may be constants, scalar variable references, array element references, array names (no subscripts), expressions, or subprogram references.

If a subprogram identifier is used as an argument, the identifier is not followed by an argument string, since this argument form is only used to provide the called subroutine with a subprogram reference. In this sense, the subprogram reference is merely a name, and as such has no value associated with it.

The name of a subroutine has no bearing on the mode of its results.

RETURN Statement

The RETURN statement returns control from an external subprogram to the calling program. Thus, the last statement executed in a subprogram will be a RETURN. It need not be physically the last statement, but may appear at any point in the subprogram where it is desired to terminate execution. Any number of RETURN statements can be used.

Form	Example
RETURN	RETURN

Within a Function subprogram, the RETURN statement causes the latest value assigned to the function name to be returned, as the function value, to the expression in which the function reference appeared. In a SUBROUTINE subprogram, RETURN causes transfer to the first executable statement following the CALL statement which passed control to the subroutine.

END STATEMENT

An END statement is required in a FORTRAN program to inform the compiler that it has reached the physical end of a program.

Form	Example
END	END

If the END statement is omitted, the compiler prints the following message

PROGRAM REQUIRES AN END CARD

8. INPUT/OUTPUT STATEMENTS

Input and output statements provide the capability of communicating with devices external to the computer. Input statements enable a program to receive information from external sources for storage in memory, while output statements allow transmission of information from storage to external sources.

In the time-sharing environment, data files are normally created at the teletype (manually and from paper tape), maintained on a large disc, and printed out at the teletype. If the use of other devices, such as the printer, is required, special request should be made to the computer center.

The number of statements in SDS 940 FORTRAN IV necessary to input and output data has been reduced to four. The ACCEPT and DISPLAY statements are used for format-free I/O to and from the teletype. The READ and WRITE statements are used for conventional formatted I/O to or from any user file. (In addition, the OPEN and CLOSE statements control availability of files during program execution.)

INPUT/OUTPUT LISTS

All input/output statements include a list that defines data to be processed by the statement. Input lists specify variables to which incoming data are to be assigned. Output lists specify expressions whose values are to be transmitted to an external device.

Simple List Items

Simple list items appear in the form

Form	Examples	
	Input Lists	Output Lists
$e_1, e_2, e_3, \ldots, e_n$	A	E
	Q(25, L)	I(J(H), N)
	RY, Y(U, B), XYZ	753820, T**5, B/3. +Y

where the e_i are expressions of any mode for output lists and variable references of any type for input lists. The comma characters must be present. The items in the list must be given in the same order as their corresponding values actually exist on the input medium or will exist on the output medium.

On input, values of variables read early in a list may be used in subscript or control expressions for variables occurring later in the list. For example, the list

K, A(K + 1)

may be used to read in a value for K and then to use that value in the subscript of variable A.

Do-Implied List Items

Indexing similar to that used in DO statements is allowed in input/output lists for handling array variables. This technique is often referred to as "self-indexing" or an "implied DO-loop". The variables to be transmitted are listed, followed by a control expression, and the whole is enclosed in parentheses to act as a single item of the list. The general form of the control expression is

$v = e_1, e_2, e_3$

where

v is the index variable.

e_1 is the initial value of the index variable.

e_2 is the upper limit value of the index variable.

e_3 is the increment size; it is optional and is assumed to be 1 if not given explicitly.

The range of the implied DO includes all list items within the parentheses which physically precede the control expression. A comma must separate the last variable from the control expression.

The rules of repetition are the same as for the DO statement, as illustrated in the following examples.

Examples

List Items	Equivalent To
(X(I), I = 1, 4)	X(1), X(2), X(3), X(4)
(X(D), Y(D), D = 1, 2)	X(1), Y(1), X(2), Y(2)
(G(2*N), N = 4, 0, -2)	G(8), G(4), G(0)
T, (C(P), P = 3, 5), E, L	T, C(3), C(4), C(5), E, L
((A(I, J), I = 7, 9), J = 1, 3)	A(7, 1), A(8, 1), A(9, 1), A(7, 2), A(8, 2), A(9, 2), A(7, 3), A(8, 3), A(9, 3)
(Z(I), I = 1, N)	A(1), Z(2), ..., Z(N-1), Z(N)
(R, T(I), I = 1, 2)	R, T(1), R, T(2)

Since the variable v in a DO-loop expression exists as a regular program variable (see Chapter 7) the list:

(A(K), K = 1, 5), G(K), H(L, K, 22)

is equivalent to the list:

A(1), A(2), A(3), A(4), A(5), G(6), H(L, 6, 22)

Lists may be omitted when the I/O statement refers to a FORMAT statement that contains only Hollerith specifications.

FREE FORMAT I/O

The ACCEPT and DISPLAY statements are designed to relieve users of the burden of providing formats when explicit format control is not required. The device accessed is always the teletype.

ACCEPT Statement

The ACCEPT statement is used to read values from the teletype. When this statement is executed, the computer waits for data to be input by the programmer, and assigns these values to variables in a list.

Form	Example
ACCEPT list	ACCEPT A, (B(I), I = 1, 3), CHECK

Brackets are optional.

The type of the list variable determines the form in which conversion from external to internal form takes place. The rules governing storage are similar to those for assignment statements (see Chapter 6). For example, if the type of the list variable is integer and a floating point number is input, the number will be truncated to an integer prior to storing.

If the type of the list variable is complex, two numbers will be demanded. The first number read will be assigned to the real part, the second to the imaginary part. If the type of the list variable is logical, the first character read must be a T for true or F for false, and any remaining characters up to the delimiter (see below) will be discarded.

Values input to the ACCEPT must be separated from each other by a space, comma, line feed or carriage return. Values will be demanded until the variable list is satisfied. Then either a carriage return or line feed may be given to cause control to return to the program.

Successive delimiters with no values typed in between them will cause zero values to be assigned to the appropriate variables in the list.

In the example above, the user could input the following:

1 (LF)

-3, 10E6, -16.6 (LF)

T (RET)

Assuming that the variable CHECK was data typed previously as LOGICAL in a Type statement, these values would be interpreted as 1., -3., 10 x 10^6, -16.6, and "true", respectively.

DISPLAY Statement

The DISPLAY statement is used to print data at the teletype. The data is transmitted as values of expressions in a list.

Form	Examples
DISPLAY list	DISPLAY [A, B+1] DISPLAY 'PRICE = $", X

Brackets are optional.

The mode of a list expression determines the manner in which its value is converted from internal to external form. In general, values are output to maximum accuracy, with all leading spaces and trailing zeros suppressed. Values are separated by three spaces and are output until the list is satisfied.

A real value is output with or without an exponent depending on the magnitude of the number, and according to the same rules as govern G-type output.

Complex values will be output as two real variables separated by commas. For example,

1.23, 5.65

If an expression is logical, the output will be a T for true or an F for false.

Character strings may be output by enclosing the string within single and double primes. Thus, if X = 100, then

DISPLAY 'PRICES = $", X

will produce the line

PRICES = $bbb100.

where b denotes blank.

The system automatically checks to see if sufficient room is available on the current teletype line (72 positions) for data. When sufficient room does not exist, a carriage return is issued, and the next value is output at the beginning of the next line. When the list has been satisfied, a carriage return is automatically issued.

FORMATTED INPUT/OUTPUT

When the programmer wishes to have explicit control over the form in which data are transmitted, formatted input/output statements are used. In general, SDS 940 FORTRAN IV formatted input/output follows conventional FORTRAN rules (see FORMAT Statement later in this chapter).

All data files (other than the teletype, which may also be referenced as a file) must be explicitly opened and closed by the two statements discussed below.

OPEN Statement

The OPEN statement makes a data file available for input or output. A program may have 3 files, in addition to the teletype, open at the same time. Any file that is to be opened must have been defined in the user's file directory prior to execution of the OPEN statement.[†]

Form	Examples
OPEN (number,/name/, use, type)	OPEN (2,/IN/, INPUT, BINARY) OPEN (3,/FILE1/, OUTPUT)

where

number is a file number that must be assigned a value between 0 and 4 inclusive. The numbers 2, 3, and 4 are used for disc files. The numbers 0 and 1 are reserved for teletype input and output respectively; however, since the teletype is always open, an OPEN statement for the teletype is redundant. The file number may appear as an expression.

name is the file name as it appears in the user's file directory. Disc file names must be enclosed in slashes. (See SDS 940 Terminal User's Guide for details concerning creation of files.)

use is either INPUT or OUTPUT.

type is either SYMBOLIC or BINARY. Type is optional; if not given, SYMBOLIC is assumed. Parentheses are required.

CLOSE Statement

The CLOSE statement closes the designated file, making it unavailable for input or output until reopened.

Form	Example
CLOSE (file number)	CLOSE (3)

where the definition of a file number is the same as for the OPEN statement. Parentheses are required.

When a file is closed and then reopened, the next I/O statement to reference it will access the beginning of the file.

All files are automatically closed upon completion of program execution.

READ Statement

The READ statement is used to input data from a file and store the data as values of the variables in a list.

Form	Examples
READ (n, f) list	READ (3, 50)A, B, (C(I), I = 1, 10) READ (4, 'I10")N
READ [n] list	READ [4] X, Y, ZED

where n is a file number and f is a format statement label or a character string enclosed in single and double primes.

[†]An output file may be defined in the user's directory by writing anything on it with the Executive command COPY. Data output to the file during program execution will write over the original contents.

The file number n is assigned in the OPEN statement. Data is converted from external to internal from according to format f, which may be either the label of a format statement or a format expressed as a character string. The format reference is omitted for binary files, as in the second form given above. Note that parentheses are required for formatted symbolic files and brackets are required for binary files.

A READ statement accesses the teletype if n = 0. A physical record terminates with a carriage return. When the list is satisfied, the data is scanned until a carriage return is read.

WRITE Statement

Forms	Examples
WRITE (n, f) list	WRITE (3, 1) X, Y, Z WRITE (4, 'F10.2") A, B
WRITE [n] list	WRITE [2] (C(J), J = 1, 100)

where n is a file number and f is a format statement label or a character string enclosed in single and double primes.

The file number n is assigned in an OPEN statement. Data is converted from internal to external format according to format f, which may be either the label of a format statement or a format expressed as a character string. The format reference is omitted for binary files, as in the second form given above. Note that parentheses are required for formatted symbolic files, and brackets are required for binary files.

A WRITE accesses the teletype if n = 1. A carriage return is output after 72 characters have been typed and/or upon satisfaction of the list.

FORMAT Statement

FORMAT statements specify the conversion to be performed on data being transmitted during a formatted input/output operation. In general, conversion performed during output is the reverse of conversion performed in an input operation. FORMAT statements are expressed as

Form	Example
FORMAT ($f_1, f_2, f_3, \ldots, f_n$)	FORMAT (F6.1, E13.1)

where the f_i are field specifications to be described in the following pages.

Each FORMAT statement must be labeled so that references may be made to it by formatted input/output statements. In addition, an entire FORMAT (the parentheses characters and the items they enclose but not the word FORMAT) may be stored in an array variable; in this case the array itself is referenced by the input/output statement. (See "FORMATs Stored in Arrays".)

FIELD SPECIFICATIONS

Field specifications describe the kind or type of conversion to be performed, specific data to be generated, scaling of data values, and editing to be executed. Each integer, real, double precision, or logical datum appearing in an input/output list is processed by a single field specification while complex data are operated on by two consecutive field specifications.

Field specifications may be any of the following forms:

```
rFw.d    rGw.d    rAw    rX
rEw.d    rIw      nHs    iP
rJc.d    rOw      $s$    r/
rDw.d    rLw      's"    Z
```

where:

1. The characters F, E, J, D, G, I, O, L, A, H, $, single prime (') and double prime ("), X, P, slash (/) and Z define the type of conversion, data generation, scaling, editing, and FORMAT control.

2. r is an optional, unsigned decimal integer that indicates that the specification is to be repeated r times; thus 3I6 is equivalent to I6, I6, I6.

3. c, for the J specification, is an unsigned decimal integer and specifies the number of digits appearing before the decimal point.

4. w is an unsigned decimal integer that defines width in characters (including digits, decimal points, and algebraic signs) of the external representation of the data being processed.

5. d, for F, E, D, and input G specifications, is an unsigned decimal integer and specifies the number of fractional digits appearing in the magnitude portion of the external field.

6. n is an unsigned decimal integer that defines the number of characters being processed.

7. s is a string of the characters acceptable to the FORTRAN IV processor.

8. i is a signed, decimal integer. The function of i is described under the P specification.

F Conversion. Integer, real, double precision, or either part of complex data may be processed by this form of conversion. Double precision values are converted with full precision if sufficient width is specified by w, and the value of d allows for the appropriate number of digits in the fractional portion of the field.

Forms	Examples
Fw.d	F6.1
rFw.d	2F10.4

Output. Internal values are converted to real constants, truncated at d decimal places with an overall length of w. The field is right justified with as many leading blank characters as necessary. Negative values are preceded with a minus sign. Consequently, for the specification F11.4

273.4	is converted to	273.4000
7	is converted to	7.0000
-.003	is converted to	-.0030
-442.30416	is converted to	-442.3041

If a value requires more positions than are allowed by the magnitude of w, an asterisk will be output, followed by the sign and as many significant digits as possible. In order to insure that such a loss of digits does not occur, the following relation must hold true:

$$w \geq d + 2 + n$$

where n is the number of digits to the left of the decimal point.

Input. Input strings may take any of the integer, real, or double precision forms discussed under "Numeric Input Strings". Each string will be a length w with d characters in the fractional portion of the value. If a decimal point character is present in the input string, the value of d is ignored, and the number of digits in the fractional portion of the value will be explicitly defined by that decimal point character.

For the specification F10.3

33	is converted to	.033	.34562	is converted to	.34562
802142	is converted to	802.142	-7.001	is converted to	-7.001

E Conversion. Integer, real, double precision, or either part of complex data may be processed by this form of conversion. Double precision values are converted with full precision if sufficient width is specified by w and the value of d allows for the appropriate number of digits in the fractional portion of the field.

Forms	Examples
Ew.d	E13.1
rEw.d	10E9.6

Output. Internal values are converted to real constants of the form

.ddd...dE±ee

where ddd...d represents d digits, while ±ee is interpreted as a multiplier of the form

$10^{\pm ee}$

Internal values are truncated to d digits, and negative values are preceded by a minus sign. The external field is right justified and preceded by the appropriate number of blank characters. The following are examples for the specification E14.8:

90.4450	is converted to	.90445000E.02
-435739015.	is converted to	-.43573901E+09
.000375	is converted to	.37500000E-03
-1	is converted to	-.10000000E+01
.2	is converted to	.20000000E+00
0.0	is converted to	.00000000E+01

The field width is counted from the right and includes the exponent digits, the sign (minus or space), the letter E, the magnitude digits, the decimal point, and the sign of the value (minus or space). If a width specification is of insufficient magnitude to allow expression of an entire value, an asterisk will be output, followed by the sign, decimal point, E character, sign of the multiplier, and as many significant digits as possible. To prevent a loss of this nature, it is necessary to insure that the relation

$w \geq d + 6$

is present in the field specification.

Input. Input strings may take any of the integer, real, or double precision forms discussed under "Numeric Input Strings".

Examples:

Value	Specification	Converted to
10.3456E03	E10.2	10345.6
-113409E2	E11.6	-11.340900
-409385E-03	E11.2	-4.09
849935E-02	E10.5	.08499

First, the decimal point is positioned according to the specification; then, the value of the exponent is applied to determine the actual position of the decimal point. In the second example, -113409E2 with a specification of E11.6 is interpreted as -.113409E02 which, when evaluated (i.e., $-.113409 \times 10^2$), becomes -11.340900.

J Conversion. Conversion of this type is similar to E conversion, except that c specifies the number of digits before the decimal point.

Forms	Examples
Jc.d	J1.2
rJc.d	6J8.1

Field width is defined by the relation

w = d + c + 6

Output. Internal values are truncated to d digits and negative values are preceded by a minus sign. The following are examples for the specification J3.4:

123.0	is converted to	123.0000E+00
9.64931	is converted to	964.9310E-02
-.001	is converted to	-100.0000E-05

Input. On input, conversion is identical to E-type conversion.

D Conversion. Conversion of this type is similar to E conversion, with the exception that for output, the character D will be present instead of the character E.

Forms	Examples
Dw.d	D11.2
rDw.d	3D6.4

For example,

for E12.6, -667.334 is converted to -.667334E+03

and

for D12.6, -667.334 is converted to -.667334D+03

G Conversion. Integer, real, double precision, or either part of complex data may be processed by this form of conversion. Double precision values are converted with full precision if the magnitude of w is adequate, and the number of significant digits defined by the value of d is sufficient to allow complete expression of the data value.

Forms	Examples
Gw.d	G2.3
rGW.d	3G4.5

Output. The purpose of the G format for output is to express numbers in a form which is most natural; that is, they are expressed in the form that is normally used for values of the corresponding magnitude.

Internal values are converted to real constants. The form of the constants is dependent upon the magnitude of the data, and conversion is either E- or F-type as indicated below, where M represents the magnitude of the data:

For $10^{i-1} \leq M < 10^{i}$

conversion will be

Fn.m

when $0 \le i < d$ and otherwise will be

Ew.d

where $n = w - 4$ and $m = d - i$.

Values converted with the F specification are followed by four blank characters in the external character string, and any non-zero scale factor in effect (see "P Specification") during F conversion is ignored. Non-zero scale factors in effect during E conversion are utilized.

The following are examples for the specification G9.2:

-1.773 is converted to -1.7ƀƀƀƀ

.133 is converted to .13ƀƀƀƀ

532. is converted to 532.00

-.0947 is converted to -9.47E-02

-.0996 is converted to -.99E-01

where ƀ represents the character blank.

If the magnitude of the width w is insufficient to allow representation of the data value, digits are listed as in E and F conversion.

Input. On input, processing is identical to F conversion.

I Conversion. Integer, real, double precision, or either part of complex data may be processed by this form of conversion. If the width specification w is of sufficient magnitude, real and double precision values are converted in full precision.

Forms	Examples
Iw	I5
rIw	7I6

Output. Internal values are converted to integer constants. Real and double precision data are truncated to integer values; however, the integers may contain as many digits as are specified by w.

Negative values are preceded by a minus sign, and the field will be right justified and preceded by the appropriate number of blank characters. The specification I6 implies that

273.4 is converted to 273

7 is converted to 7

-.003 is converted to 0

-44205.965 is converted to -44205

If the magnitude of data requires more positions than permitted by the value of the width w, an asterisk will be output, followed by the sign and as many significant digits as possible.

Input. On input, conversion is identical to F-type processing except that fractional portions of a value are lost through truncation.

O Conversion. O conversion is used to process octal values.

Forms	Examples
Ow	O8
rOw	3O8

Output. Internal binary word values, with no regard to data type, are converted to their octal equivalents. In order to fully represent each data type, the following requirements are placed on the value of the width w:

1. Double precision data require 24 digits

2. Real and integer data require 16 digits

Note that real data include either part of complex data and Hollerith information. Logical data cannot be output with an O conversion.

Example

Data Values	Internal Binary	rOw	External Octal
1	010000000000000000000000 000000000000000000000001	O16	2000000000000001
5.0	010100000000000000000000 000000000000000000000011	O16	2400000000000003
1D0	011001100110011001100110 011001100110011001100110 011001100110011111111101	O24	31463146 31463146 31463775
HOL	001010000010111100101100	O8	12027454

Whenever the magnitude of w is insufficient for the complete expression of a value, digits will be lost from the least significant portion of the field.

If w is of a magnitude greater than that necessary to express the octal representation of the data, the field in the external string will be right justified and preceded by the appropriate number of zero characters.

Input. External fields processed by O conversion may contain only strings of octal digits and blank characters. If a field contains other than one of the above, an error occurs.

Conversion begins with the first character in the string, including blanks. Blank characters are treated as if they were zero characters. Thus

    ƀ35ƀƀ71ƀ

is equivalent to

    03500710

for the specification O8, where ƀ represents the character blank. Also, fields that contain nothing but blank characters are assumed to have the value zero.

Fields that contain more significant digits than required by the corresponding list items lose digits from the least significant portion of the field. For instance, if the list item is integer, and the input specification used is O24 then

    123456700765432112345670

is converted to

    1234567007654321

L Conversion. Only logical data may be processed with this form of conversion; any other data type causes an error to occur.

Forms	Examples
Lw	L1
rLw	5L4

Output. Logical values are converted to either TRUE or FALSE for the values "true" and "false" respectively. If the field width will not contain the full word, either a T or an F character is output. The T and F characters are preceded by w-1 blank characters. For example, using the specification L4.

.TRUE. is converted to TRUE

.FALSE. is converted to ƀƀƀF

where b represents the character blank.

Input. The first non-blank character in the input string must be either a T or an F character; any other character appearing at the first non-blank character causes an error to occur. The occurrence of a T or an F character causes the corresponding list item to be assigned the values "true" or "false", respectively.

Thus,

TRUE and FALSE

are valid input strings. Characters falling between the T and F characters and the right-hand boundary of the external field are ignored. Fields consisting of only blank characters cause an error condition.

A Conversion. A conversion is used to process character strings.

Forms	Examples
Aw	A6
rAw	4A3

Output. Internal binary values are converted to character values at the rate of eight binary digits per character. The most significant digits are converted first; i.e., conversion is from left to right. Internal values are processed in the following manner:

Data Type	Internal Binary	rAw	External String
integer	001010010010111000110100 000000000000000000000000 (1222706400000000 octal)	A3 A2	INT IN
real	001100100010010100100001 001011000011101100011000 (1442244113035430 octal)	A6 A11	REAL 8 ƀƀƀƀƀREAL[8
double precision	001001000010111100110101 001000100010110000100101 000000000001110000011110 (110274651042604500016036 octal)	A9 A12	DOUBLEƀ<> ƀƀƀDOUBLEƀ<>

When the magnitude of w does not provide for enough positions to express the data value completely (6 for real or integer, 9 for double precision data), the external field is shortened from the right or least significant portion. When w has a value greater than necessary, the external character string is preceded by the appropriate number of blank characters.

This type of conversion is normally used to output Hollerith information which has been placed in storage.

Input. Hollerith input may be stored in real variables only. When the value of w is less than 6, the list item is filled with the w characters in the most significant positions, and the remainder of the positions are filled with blank characters. Consequently, if the field specification is A4,

UVWX is converted to UVWXƀƀ

where ƀ represents the character blank.

When the width w is larger than 6 an error condition occurs.

A general rule for this type of conversion is that internal values are considered to be left justified, while external fields are considered to be right justified.

H Conversion. H conversion takes the form

Form	Example
nHs	5HTOTAL

Output. The n characters in the string s are transmitted to the external medium. For instance:

Specification	External String
1HE	E
8HƀƀVALUE:	ƀƀVALUE:
5H$3.95	$3.95
9HX(2, 5)ƀ =ƀ	X(2, 5) ƀ =ƀ

where ƀ represents the character blank.

Care should be taken that the character string s contains exactly n characters, so that the desired external field will be created, and so that characters from other field specifications are not used as part of the string.

Input. The n characters in the string s are replaced by the next n characters from the input record. This replacement occurs as shown in the following examples:

Specification	Input String	Resultant Specification
3H123	ABC	3HABC
10HNOWƀISƀTHE	ƀTIMEƀFORƀ	10HƀTIMEƀFORƀ
5HTRUEƀ	FALSE	5HFALSE
6Hƀƀƀƀƀƀ	RANDOM	6HRANDOM

where ƀ represents the character blank. This feature can be used to change titles, dates, column headings, etc., that are to appear on an output record generated by the H specification.

If n is not present or is equal to zero, an error condition occurs.

$ Conversion. $ conversion is similar to H conversion, except that a character count is not required.

Form	Example
s	$TEXT$

The string s may contain any character other than a dollar sign character ($) and the control characters given in Chapter 2.

Output. The string s is transmitted to the external device in a manner similar to that for H conversion. Thus,

    $DOLLAR SIGN$

is output as the string

    DOLLAR SIGN

Input. The characters appearing between the dollar sign characters are replaced by the same number of characters taken sequentially from the input string. Therefore, for the input string

    MATRIX

and the specification

    $VECTOR$

the resultant specification is

    $MATRIX$

'-" Conversion. This '-" conversion is identical to $ conversion, with the exception that dollar sign characters may be present in the string s.

Form	Examples
's"	'TWAS BRILLIG AND THE SLITHY TOVES... " 'WHAT, 'ME WORRY?""

The prime and double prime characters may be used within the string, but they must occur in pairs as they denote strings within strings.

Blank characters in FORMAT statements are significant only in H, $, and '-" specifications.

X Specifications. The X specification causes no conversion to occur. Instead, it causes i positions of an external field to be "skipped". i must be positive.

Form	Example
iX	3X

Output. The next i positions of the output record will be blank characters. In other words, a field of i blank characters will be created. The specifications

    $WXYZ$, 4X, 'IJKL"

cause the external string

    WXYZƀƀƀƀIJKL

to be generated, where b represents the character blank.

Input. The next i characters from the input string are ignored. For example, with the specifications

    F5.3, 6X, I3

and the input string

    76.42IGNORE597

the characters IGNORE will not be processed.

P Specifications. P specifications cause the value of the scale factor to be set to i.

Form	Examples
iP	2P -6P

where the scale factor is treated as a multiplier, of the forms

10^i for output

and

10^{-i} for input

At the beginning of each formatted input/output operation, before any processing occurs, the scale factor is assigned a value of zero. Any number of P specifications may be present in a FORMAT statement, thereby causing the value of the scale factor to be changed several times during a formatted input/output operation. If a FORMAT is restarted within a single operation due to the number of items in a list, the value of the scale factor is not reset to zero.

Scale factors are effective only with F, E, and D conversions, input G conversions, and E-type output for G conversions.

Output. The value of a list item is scaled by the multiplier 10^i. This scaling causes the decimal point character which appears in the output string to be shifted i places. For E- and D-type conversions, the exponent fields are reduced by the value of i.

Thus, for the value .234, various specifications and their results are:

Specification	Output
F5.3	.234
2P, F6.3	23.400
-2P, F5.3	.002
E9.3	.234E+00
2P, E11.3	23.400E-02
-2P, E9.3	.002E+02

The results for D conversion and E-type output for output G conversions would be similar to those for E conversion.

Input. During F, E, D, and G input conversions, if the input string contains an exponent field, the scale factor has no effect. However, when the input string does not contain an exponent field, the value of the external field is scaled by 10^{-i}. The following examples indicate the effect of scaling during an input operation:

External Field	Scale Factor	Effective Value
-71.436	0P	-71.436
	3P	-.071436
	-1P	-714.36
-71.436E+00	3P	-71.436
	1P	-71.436

Once a scale factor has been established during an input/output operation, it remains in effect throughout the operation unless redefined by an additional P specification. Thus for the list

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O

and the FORMAT

FORMAT (F4.3, 2P, I2, E5.2)

A is converted with a zero scale factor and the specification F4.3; B is converted with the specification I2; C is converted with a scale factor of 2 and the specification E5.2. Since there are no more specifications and yet the list is not exhausted, the FORMAT is re-scanned for the next set of three list items (see "FORMAT and List Interfacing"). Thus, while A is processed with a zero scale factor, D, G, J, and M are processed with a scale factor of 2, although they are all converted with the specification F4.3.

When i is not specified, its value is assumed to be zero. Thus P is equivalent to 0P.

/Specifications. Slash (/) specifications cause another record to be processed.

Forms	Examples
/ r/	/ 2/

In the case of contiguous slash specifications (i.e., ////.../ or r/), since no conversion occurs between each of the slash specifications, records are ignored during input (scanned to a carriage return), and empty records are generated during output operations. The same condition can occur when a slash specification and either of the parenthesis characters surrounding the field specifications are contiguous.

Output. When a slash specification is encountered, the current record being processed is output and another record is begun. If no conversion has been performed when the slash is sensed, an empty record is created. (On the teletype, this would be a blank line.) The statements

WRITE (4, 10) A, B

10 FORMATS (F5.3, //, I13)

are processed in the following manner:

1. A record is begun, and A is converted via the specification F5.3.

2. The first slash is encountered, the record containing the external representation of A is terminated, and another record is begun.

3. The second slash causes termination of the second record, and a third record is started. Since no conversion occurred between the terminations of the first and second records, the second record was empty.

4. The value B is converted with the I13 specification, the closing right parenthesis character is encountered, and the third record is terminated.

If a third item C were added to the output list, as in

WRITE (4, 10) A, B, C

the following additional steps would occur:

5. A fourth record is begun, and C is converted using the specification F5.3.

6. The first slash is re-encountered, the fourth record is terminated, and a fifth record is begun.

7. Again, the second slash is processed; the fifth record, which is empty, is terminated; and the sixth record is started.

8. Since there are no more list items, the specification I13 is not processed, a termination occurs, and the final or sixth record, which is also empty, is output.

The original FORMAT statement could also have been written as

```
10 FORMAT (F5.3, 2/I13)
```

or

```
10 FORMAT (F.3, 2/, I13)
```

with the identical effect.

The two statements

```
WRITE (3,4) X
4 FORMAT (3/E6.4/)
```

cause the generation of three empty records, followed by a record containing the value of X, converted by the specification E6.4, followed by another empty record.

Input. The effect of slash specifications during input is similar to the effect for output, except that for input, records are ignored where empty records would be created during output. For example, the statements

```
WRITE (3,4) X
4 FORMAT (3/E6.4)
```

cause three records to be bypassed (i.e., 3 carriage returns to be read), a value from the fourth record to be converted with the specification E6.4 and assigned to X, and a fifth record to be bypassed. This means that, as with the last example for output, records created with a FORMAT statement containing slash specifications can be input by use of the identical FORMAT statement.

Z Specifications. The Z specification takes the form

Form	Example
Z	Z

Output. On output, the Z specification causes the suppression of the terminal carriage return normally issued upon termination of output. This feature is useful when outputting requests for input from the teletype. For example,

```
WRITE (1, 10)
10 FORMAT ($A=$Z)
```

will cause

```
A=
```

to be printed at the teletype with the carriage positioned after the last character typed. If the next statement executed is an ACCEPT, the user's input will be typed on the same line.

Input. This specification is ignored on input.

Repetition of Field Specifications. Within a FORMAT statement, any number of field specifications may be repeated by enclosing them within parentheses, preceded by a repeat count, in the following form:

Form	Example
$r(f_1, f_2, f_3, \ldots, f_n)$	(3(A4, F4.2, 3X), 3I)

where r is the repeat count and the f_i are specifications. Thus the statement

```
5 FORMAT (3(A4, F4.2, 3X), 3I)
```

is equivalent to

```
5 FORMAT (A4, F4.2, 3X, A4, F4.2, 3X, A4, F4.2, 3X, 3I)
```

The repetition count may be any number up to $2^{24}-1$.

During input/output processing, each repetitive specification is exhausted in turn, as is each singular specification.

Examples:

```
  34 FORMAT (4X, 2(A8, I1, 7G9.3), I4, 3(L5))
1125 FORMAT (/, A4, F9.7, 5(E14.8, 2/), E14.8)
   8 FORMAT (7(I8, 2(3X, F12.9), F12.9), A16)
```

In the last example above, repetitions have been nested. Nesting of this type is permissible to a depth of ten levels.

NUMERIC INPUT STRINGS

Input strings processed by F, E, D, J, G, and I conversions may take any of the following forms:

Forms	Examples
±n	2500
±n.m	2500.0
±n±e	25 + 02
±n.m±e	25.0 + 02
+nE+e	25E + 02
±n.mE±e	25.0E + 02

where n, m, and e are strings of decimal digits or blank characters; plus sign characters are optional except prior to e when the character E is not present; and the decimal point and E characters must be present in that form. The character D may be substituted for the E character with no change in meaning or value.

Blank characters in the strings n, m, and e are treated as zero characters, as are n, m, and e if they are empty strings.

When conversion is via an I specification, fractional portions of a value are lost through truncation.

In all cases, conversion begins with the first non-blank character in the field, and blank characters falling between the E (or D) character and the exponent field are ignored.

TERMINATION OF INPUT STRINGS

Normally a READ statement inputs the exact number of characters specified in the field specification. However, a field may be terminated short by an I^C. Note that the input string will be considered left justified and the field will be zero filled. The next field begins with the character following the line feed. (LF) will fill the same purpose, except that if the input media is the teletype, spaces equal to the number of characters needed to satisfy the format will be echoed to the teletype.

For example, using the specification 3F7.3, the input string

3450(LF)88412(LF)33.21(LF)

is equivalent to

3.45 88.412 33.21

If I^c is substituted for (LF), the input string is equivalent to

3450000884120033.2100

(RET) in a field will be ignored until the list is satisfied, providing a means of automatic continuation from one physical line to the next on the teletype.

If a colon is used to terminate an input string, enough blanks will be appended to the current input line to give it a total length of 80 characters. If more than 80 characters have been read then the colon is read as any other character. This feature is provided for users with decks of data cards which have been converted to disc files at the computer center. The card-to-disc file routine places a colon after the last significant character followed by a carriage return unless the last significant character is in column 80. When column 80 is non-blank, the colon is omitted. This feature allows input data to be easily edited in QED.

FORMAT AND LIST INTERFACING

Formatted input/output operations are controlled by the FORMAT requested by each READ or WRITE statement. Each time a formatted READ or WRITE statement is executed, control is passed to the FORMAT processor, which operates in the following manner:

1. When control is initially received, the processor prepares to read a new record or line, or to construct a new output record or line.

2. Subsequent records are started only after a slash specification has been processed (and the preceding record has been terminated) or the final right parenthesis of the FORMAT has been sensed, or the maximum number of characters for a teletype line has been output.

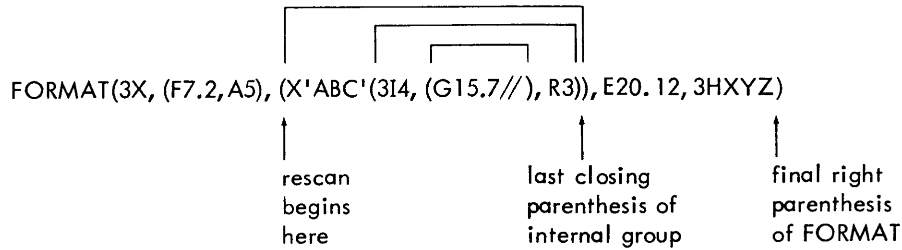
3. During an input operation, processing of an input record is terminated whenever a slash specification or the final right parenthesis of the FORMAT is sensed, or when the FORMAT processor requests an item from the list and no list items remain to be processed. Construction of an output record terminates, and the record is written on occurrence of the same conditions.

4. Every time a conversion specification (i.e., F, E, J, D, G, I, O, L, or A specification) is to be processed, the FORMAT processor requests a list item. If one or more items remain in the list, the processor performs the appropriate conversion and proceeds with the next field specification. (If conversion is not possible because of a conflict between a specification and a data type, an error occurs.) If the next field specification is one which does not require a list item (i.e., H, $, '-", Z, X, P, or /), it is processed whether or not another list item exists. When there are no list items to be processed, the current record is terminated and control is passed to the statement following the READ or WRITE statement that initiated the input/output operation.

5. When the final right parenthesis of a FORMAT statement is encountered by the FORMAT processor, a test is made to determine if all list items have been processed. If the list has been exhausted, the current record is terminated, and control is passed to the statement following the READ or WRITE statement that initiated the input/output operation. However, if another list item is present, an additional record is begun, and the FORMAT statement is rescanned. The rescan takes place as follows:

   a. If there are no parenthesized groups of specifications within the FORMAT statement, the entire FORMAT is rescanned.

   b. If, however, one or more parenthesized groups do appear, the rescan is started with the group whose right parenthesis was the last one encountered prior to the final right parenthesis of the FORMAT statement. In the following example, the rescan begins at the point indicated:



c. If the group at which the rescan begins has a repeat count (r) in front of it, this value is used again for each rescan.

6. Each list item to be converted is processed by one specification or one iteration of a repeated specification, with the exception of complex data which are processed by two such specifications.

7. Each READ or WRITE statement containing a non-empty list must refer to a FORMAT statement that contains at least one conversion specification (see Step 4 above). If this condition is not met, the FORMAT statement will be processed, but an error will occur.

FORMATS STORED IN ARRAYS

A FORMAT, including the beginning left parenthesis character, the final right parenthesis character, and the specifications enclosed therein, (but not the word FORMAT), may be stored in an array variable. The FORMAT must be stored as a Hollerith constant (i.e., a string of characters) by use of either an input statement or an assignment statement.

READ or WRITE statements that refer to a FORMAT stored in an array must reference only the identifier of the array, with no subscription. For example

```
WRITE (4,R) E, F, G
```

refers to a FORMAT stored in array R.

If the variable Z is a REAL array, and the string to be stored is (F8.5, 4HNODE, I3), two methods may be used:

1. The string may be read in at execution time. For example

```
   READ  (M,90)  (Z(I),I=1,3)
90 FORMAT  (3A6)
```

2. Assignment statements may be used to achieve the same effect. For example

```
Z(1) = 6H(F8.5,
Z(2) = 6H4HNODE
Z(3) = 6H,I3)bb
```

Care must be taken when storing into an array a FORMAT containing specification of the nHs, s, and 's" forms. In these cases, all characters in the string s, including blank characters, are significant. For example, if an A4 format had been used to read in the string in the example above, the following results would have occurred:

Element	Storage after READ
Z(1)	(F8.bb
Z(2)	5,4Hbb

Element	Storage after READ
Z(3)	NODEbb
Z(4)	, I3)bb

which is not the desired result, since it is equivalent to the FORMAT:

(F8.5, 4HbbNODE, I3)

where b represents the character blank.

Even though a FORMAT may be quite short, it must be stored in an array rather than a scalar variable.

Using the teletype as the input file, this feature may be used to good advantage during on-line checkout. Programs may be tested with minimal formats, but once a program is operational, the output format may be expanded to any desired level. Or, during checkout, a part of the output may be suppressed altogether with F0.0, I0, or E0.0 specifications. On the other hand, it is much easier to design, test, and modify complex formats while actually observing program output on-line.

9. DECLARATION STATEMENTS

Declaration statements are used to define the data type of a variable or function subprogram, the dimensions of an array variable, and the initial values of variable data, and to provide other similar information to the processor. Several FORTRAN IV statements are used solely for the purpose of supplying the system with declarative information. These statements are primarily concerned with the interpretation of identifiers occurring in the source program.

CLASSIFICATION OF IDENTIFIERS

Each identifier appearing in a source program is classified as the language element it identifies. Three main classes are recognized:

scalar identifiers

array identifiers

subprogram identifiers.

The category in which an identifier is placed, and the data type (if any) associated with it are dependent upon the context in which the identifier is initially defined. This definition is a declaration, explicit or implicit, of the way in which the identifier is to be categorized throughout the remainder of the program.

Implicit Declarations

Unless specifically declared to be in a particular category or type, identifiers which appear in executable or DATA statements are implicitly classified according to the following set of rules:

1. Any identifier appearing in a CALL statement as the call subprogram is a subprogram identifier. For example,

   CALL ERR or CALL NIX [R, V]

2. An identifier (other than defined in paragraph 1) that is followed by an argument list enclosed in brackets, such as A [T, ALPHA, B+C] is

   a. A statement function definition if it appears in the manner discussed under "Statement Functions" in Chapter 10.

b. A function subprogram reference if it appears in an expression. This does not apply to identifiers appearing to the left of a replacement operator (=).

c. An error if it appears to the left of a replacement operator in any statement other than a statement function definition.

3. An identifier that is not followed by an expression list enclosed in parentheses is defined as a scalar variable.

4. When applicable, the data type associated with an identifier is integer if the identifier begins with the letter I, J, K, L, M, or N. If the identifier begins with any other letter, its type is real.

5. An identifier that appears in a non-executable statement, but never in an executable or DATA statement, is implicitly classified after all statements have been processed. Classification is in accordance with the previous set of rules and depends upon the classification defined by the non-executable statement in which the identifier appears.

Explicit Declarations

All classifications of identifiers other than those discussed in the previous section require explicit definition. Explicit definitions and declarations include:

- array declarations
- type statement declarations
- subprogram definitions

ARRAY DECLARATIONS

Array declarations explicitly define an identifier as the name of an array.

Form	Examples
$v(d_1, d_2, d_3, \ldots, d_n)$	ARRAY (4:9, 15, 0:1, -20:20) CUBE (-10:-1, 5, 32) PLANE (-999:0, 1:450) LINE (140) X(1)

where v is the identifier, n is the number of dimensions associated with the array, and the d_i define the range of the corresponding dimensions.

Each d_i may take the forms:

r_u

or

$r_o:r_u$

where r_u is an integer that defines the upper bound of the dimension range, and r_o is an integer that defines the lower bound of the dimension range.

In the first form, the lower bound is assumed to be 1, and the upper bound must be positive. For example,

```
ARRAY(10)
```

defines ARRAY to be a one-dimensional array, with a range which has 1 as its lower bound and 10 as its upper, for a maximum of 10 elements.

In the second form, both the upper and lower bounds may be positive, negative, or zero valued as long as the value of the upper bound is greater than or equal to the value of the lower bound.

In the first example given above, ARRAY is defined as a 4-dimensional array. The first dimension has a range of 4 to 9, the second of 1 to 15, and so on.

Array Storage

Although an array may have several dimensions, it is placed in storage as a linear string. This string contains the array elements in sequence (from low address storage toward high address storage) so that the leftmost dimension varies with the highest frequency, the next leftmost dimension varies with next highest frequency, and so forth. Thus, a two dimensional array would be stored "column-wise", i.e., with the row subscripts varying most frequently.

Example

array A(3, 3, 3)		array B(-3:1, 0:4)	
Item	Element	Item	Element
1	A(1, 1, 1)	1	B(-3, 0)
2	A(2, 1, 1)	2	B(-2, 0)
3	A(3, 1, 1)	3	B(-1, 0)
4	A(1, 2, 1)	4	B(0, 0)
5	A(2, 2, 1)	5	B(1, 0)
6	A(3, 2, 1)	6	B(-3, 1)
7	A(1, 3, 1)	7	B(-2, 1)
8	A(2, 3, 1)	8	B(-1, 1)
9	A(3, 3, 1)	9	B(0, 1)
10	A(1, 1, 2)	10	B(1, 1)
11	A(2, 1, 2)	11	V(-3, 2)
12	A(3, 1, 2)	12	B(-2, 2)
13	A(1, 2, 2)	13	B(-1, 2)
14	A(2, 2, 2)	14	B(0, 2)
15	A(3, 2, 2)	15	B(1, 2)
16	A(1, 3, 2)	16	B(-3, 3)
17	A(2, 3, 2)	17	B(-2, 3)
18	A(3, 3, 2)	18	B(-1, 3)
19	A(1, 1, 3)	19	B(0, 3)
20	A(2, 1, 3)	20	B(1, 3)
21	A(3, 1, 3)	21	B(-3, 4)
22	A(1; 2, 3)	22	B(-2, 4)
23	A(2, 2, 3)	23	B(-1, 4)
24	A(3, 2, 3)	24	B(0, 4)
25	A(1, 3, 3)	25	B(1, 4)
26	A(2, 3, 3)		
27	A(3, 3, 3)		

References to Array Elements

References to array elements must contain the number of subscripts that correspond to the number of dimensions declared for the array. References which contain an incorrect number of subscripts are treated as errors.

Furthermore, the value of each subscript should be within the range of the corresponding dimensions as specified in the array declaration. Otherwise the references will be treated as errors.

DIMENSION Statement

DIMENSION statements are nonexecutable statements used to define the dimensions of an array. Every array variable appearing in a source program must represent an element of an array declared in a DIMENSION statement. Any number of arrays may be dimensioned in a single DIMENSION statement.

Form	Examples
DIMENSION $s_1, s_2, \ldots, s_n$	DIMENSION D(45, -50:50, 4), Y(5000), WHTAX(0:70) DIMENSION F(2, 3, 4, 5, 6), G(3)

where the s_i are array declarations

Array declarations are discussed in detail in the previous section.

COMMON Statement

The COMMON statement is used to assign variables to a region of storage called COMMON.

Form	Example
COMMON $s_1 s_2, \ldots, s_n$	COMMON E, D(10)

where the s_i are scalar or array identifiers.

The COMMON statement specifies that the scalar and arrays indicated are to be stored in an area also available to other programs. By use of COMMON statements, a common storage area may be shared by a program and its subprograms.

Each array name that appears in a COMMON statement must also appear in a DIMENSION statement in the same program.

Quantities whose identifiers appear in COMMON statements are allocated storage in the same sequence that their identifiers appear in the COMMON statements, beginning with the first COMMON statement in the program.

Storage allocation for common quantities begins at the same location for all programs. Thus, the programmer can establish a one-to-one correspondence between the quantities of several programs even when the same quantities have different identifiers in different programs. For example, if a program contains

```
COMMON A, B, C
```

as its first COMMON statement, and the subprogram has

```
COMMON X, Y, Z
```

as its first COMMON statement, then A and X will refer to the same storage location. A similar correspondence exists for the pairs B and Y, C and Z.

Identifiers that correspond in this way must agree in mode for meaningful results.

DATA STATEMENT

A DATA statement is used to initiate variables to declared values. If a DATA statement is unlabeled, the initialization occurs during loading of an executable program and prior to execution of the program. If the DATA statement has a statement label, it is treated as a normal program statement and is executed when reached in the course of program execution.

Form	Example
DATA $d_1 d_2 d_3 \ldots d_n$	DATA X, Y, Z/1, 2, 3/

The d_i take the following form:

$k/C_1, C_2, C_3, \ldots, C_m/$

where k is a list that is similar to an input list (see Chapter 8) and the C_j are either constants or repeated groups of constants. The purpose of the statement is to cause the variables in the list k to be assigned the value of the corresponding constants in C_j.

The following rules distinguish the list k from input lists:

1. No variables may be used in subscript expressions unless they are the control variable in an implied DO-loop (i.e., v in $v=e_1, e_2, e_3$) or unless they appear earlier in the DATA statement since otherwise they have no values during initialization.[†]

2. The expressions in an implied DO-loop may contain variables only under the same condition as described in paragraph 1.[†]

3. All implied DO-loops must be enclosed in parentheses.

The C_j, which are either constants or repeated groups of constants, may take any of the forms:

c_o

$r*c_o$

$r(c_1, c_2, c_3, \ldots, c_z)$

where each c may be a constant of any type and r is an unsigned decimal integer whose value is the number of repetitions of each group. In the third form, r is optional, and if not present it has an assumed value of 1.

For example:

```
DATA X, (Y(I), I=1, 5), Z/32.5, 5*0.0, -7/R, Q/1.5E3, 123.45/
```

has the same effect as the statements

```
  X=32.5
  DO 1 I=1,5
1 Y(I)=0.0
  Z=-7
  R=1.5E3
  Q=123.45
```

except that the DATA statement is effective prior to execution of the program, since it is unlabeled. Note that the expression 5*0.0 in the above example does not mean 5 times 0, but rather five zeros.

[†] Applies only to unlabeled DATA statements.

If the data type of a constant is not the same as the data type of the variable to which it is assigned, conversion occurs according to the rules in Table 2.

The list k must specify at least as many items as are specified by the list of constants. If the list k specifies more items than the list of constants, the list of constants is repeated until all the items in the list k have been assigned values. For example, if there are 3 items in the list k, the DATA statement

```
DATA A, B, C/3*1.0/
```

is equivalent to

```
DATA A, B, C/1.0/
```

and

```
DATA.A, B, C/1.0, 2.0/
```

is equivalent to

```
A=1.0
B=2.0
C=1.0
```

Variables of complex data type that appear in the list k require two constants per datum for initialization. The first of the two constants initializes the real part, the second and imaginary part of the complex datum. Two constants used for this purpose may be written as:

(c_1, c_2)

which is a repeated group of two constants with an implied repeat count of 1(i.e., the same as $1(c_1, c_2)$). Consequently,

```
COMPLEX T
REAL R, S
DATA R, S, T, /5, -48.3, (34, 8)
```

are equivalent to

```
COMPLEX T
REAL R, S, U
R = 5
S = 48.3
T = (34, 8)
```

TYPE STATEMENT

A Type statement is used to explicitly define the data type of a variable or function subprogram.

Form	Examples
$d_t v_1, v_2, \ldots, v_n$	INTEGER R, F, I, E(-5:10, 15)
	REAL M, KING(55, 55)
	LOGICAL SEQ, BOOLE(5, 5, 5, 5)

where d_t is a data type and the v_i are identifiers of variables or function subprograms, or are array declarations.

The possible data types are: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL.

Type statements may appear anywhere in a program.

10. SUBPROGRAMS

A subprogram consists of one or more lines of code executed when called upon by name by another program. The purpose of a subprogram is to make it more convenient to perform frequently occuring operations.

There are two general categories of subprograms:

1. A function subprogram is called implicitly by using its name in an expression, and it returns a single result through its identifier.

2. A subroutine subprogram is called explicitly by a CALL statement, and may return more than one value through arguments.

FUNCTION SUBPROGRAMS

Function subprograms are programmed procedures that are often used to provide solutions to mathematical functions. These subprograms are used in a manner similar to that of normal mathematical notation. For example, there is a library cosine function whose identifier is COS, thus allowing

y=cos x

to be written as

Y=COS [X]

The appearance of the identifier COS constitutes a call to the standard library subprogram COS, which is available to all SDS 940 FORTRAN IV users. Control transfers to the function, which when executed, returns a value to the function reference in the calling program. The calling program can then use this value as it would any other.

Thus function references may be used in the same manner as variable references in any expression. For example,

X=(-B+SQRT [B**2-4*A*C])/2*A

SQRT is the identifier of the square root function and [B**2-4*A*C] is the calling argument list.

There are three types of function subprograms:

Library Functions

Statement Functions

FUNCTION Subprograms[†]

Library Functions

Library ("intrinsic") functions are subprograms that evaluate commonly used mathematical functions. They are contained in the FORTRAN IV library. These functions have inherent data type classifications, as given in the table below. In the table, C signifies a complex mode; D, a double precision mode; I, an integer mode; L, a logical mode; and R, a real mode. N means number.

[†]Where the word "function" is capitalized in this test, the reference is to the specific type of function subprogram that begins with a FUNCTION statement.

Table 3. Library Functions

Library Function Names	Type of Function	Number of Arguments	Type of Arguments	Definition of Function
ABS	R	1	I, R, D	Absolute value.
AIMAG	R	1	I, R, C	Imaginary part of argument (zero if not complex) expressed as real value.
AINT	R	1	I, R, D	Integer part of argument expressed as a real value.
ALOG	R	1	I, R, D	Natural logarithm (base e).
ALOG10	R	1	I, R, D	Common logarithm (base 10).
{AMAX, AMAX1}	R	$N \geq 1$	I, R, D	Maximum value. All arguments are converted to and compared as real values.
AMAX0	R	$N \geq 1$	I, R, D	Maximum value. All arguments are converted to and compared as integer values.
{AMIN, AMIN1}	R	$N \geq 1$	I, R, D	Minimum value. All arguments are converted to and compared as real values.
AMIN0	R	$N \geq 1$	I, R, D	Minimum value. All arguments are converted to and compared as integer values.
AMOD	R	2	I, R, D	Arg_1 (mod arg_2). Evaluated as $arg_1 - arg_2$*AINT [arg_1/arg_2]; i.e., the sign is the same as arg_1.
{ATAN, ATAN2}	R	1, 2	I, R, D	Arctangent of argument. If two arguments, quadrant allocated between -pi and +pi.
CABS	R	1	I, R, C	Complex absolute value; i.e., modulus.
CATAN	C	1	I, R, C	Complex arctangent.
CCOS	C	1	I, R, C	Complex cosine.
CCOSH	C	1	I, R, C	Complex hyperbolic cosine.
CEXP	C	1	I, R, C	Complex exponential (e**arg).
CINT	C	1	I, R, C	Complex number formed by the integer values of the real and imaginary parts of argument.
CLOG	C	1	I, R, C	Complex natural logarithm (base e). Allocated between -pi and +pi.
CLOG10	C	1	I, R, C	Complex common logarithm (base 10). Allocated between -pi and +pi.
CMPLX	C	2	I, R, D	Complex number where real part = arg_1, imaginary part = arg_2; i.e., converts two real numbers to a complex number.

Table 3. Library Functions (continued)

Library Function Name	Type of Function	Number of Arguments	Type of Arguments	Definition of Function
CONJG	C	1	I, R, C	Complex conjugate. (Has no effect if argument is not complex.)
COS	R	1	I, R, D	Cosine.
COSH	R	1	I, R, D	Hyperbolic cosine.
CSIN	C	1	I, R, C	Complex sine.
CSINH	C	1	I, R, C	Complex hyperbolic sine.
CSQRT	C	1	I, R, C	Complex square root. Allocated between -pi/2 and +pi/2, i.e., the real part is positive.
DABS	D	1	I, R, D	Double precision absolute value.
{ DATAN, DATAN2 }	D	1, 2	I, R, D	Double precision arctangent. If two arguments, arctangent of arg_1/arg_2, quadrant allocated between -pi and +pi.
DBLE	D	1	I, R, D	Argument converted to double precision.
DCOS	D	1	I, R, D	Double precision cosine.
DEXP	D	1	I, R, D	Double precision exponential. (e**arg)
DIM	R	2	I, R, D	Positive difference; i.e., arg_1-AMIN [arg_1, arg_2].
DINT	D	1	I, R, D	The integer part of the argument expressed in double precision.
DLOG	D	1	I, R, D	Double precision natural logarithm (base e).
DLOG10	D	1	I, R, D	Double precision common logarithm (base 10).
{ DMAX, DMAX1 }	D	$N \geq 1$	I, R, D	Double precision maximum value. All arguments are converted to and compared as double precision values.
{ DMIN, DMIN1 }	D	$N \geq 1$	I, R, D	Double precision minimum value. All arguments are converted to and compared as double precision values.
DMOD	D	2	I, R, D	Arg_1 (mod arg_2). Evaluated as arg_1-arg_2*DINT [arg_1/arg_2]; i.e., the sign is the same as arg_1.
DSIGN	D	2	I, R, D	Magnitude of arg_1 with sign of arg_2. If arg_2 is zero, the sign is positive.
DSIN	D	1	I, R, D	Double precision sine.
DSQRT	D	1	I, R, D	Double precision square root.

Table 3. Library Functions (continued)

Library Function Name	Type of Function	Number of Arguments	Type of Arguments	Definition of Function
EXP	R	1	I, R, D	Exponential. (e**arg)
{FLOAT, SNGL}	R	1	I, R, D	Argument converted to a real value.
IABS	I	1	I, R, D	Integer absolute value.
IDIM	I	2	I, R, D	Positive difference; i.e., arg_1-MIN [arg_1, arg_2].
{INT, IFIX, IDINT}	I	1	I, R, D	Argument converted to an integer value.
ISIGN	I	2	I, R, D	Magnitude of arg_1 with sign of arg_2. If arg_2 is zero, the sign is positive.
{MAX, MAX0}	I	$N \geq 1$	I, R, D	Maximum value. All arguments are converted to and compared as integer values.
MAX1	I	$N \geq 1$	I, R, D	Maximum value. All arguments are converted to and compared as real values.
{MIN, MIN0}	I	$N \geq 1$	I, R, D	Minimum value. All arguments are converted to and compared as integer values.
MIN1	I	$N \geq 1$	I, R, D	Minimum value. All arguments are converted to and compared as real values.
MOD	I	2	I, R, D	Arg_1(mod arg_2). Evaluated as arg_1 -arg_2*INT[arg/arg_2]; i.e., the sign is the same as arg_1.
REAL	R	1	I, R, C	Argument converted to a real value. Same as FLOAT and SNGL except accepts complex arguments and returns the real part.
SIGN	R	2	I, R, D	Magnitude of arg_1 with sign of arg_2. If arg_2 is zero, the sign is positive.
SIN	R	1	I, R, D	Sine.
SINH	R	1	I, R, D	Hyperbolic sine.
SNGL	R	1	I, R, D	See FLOAT.
SQRT	R	1	I, R, D	Square root.
TANH	R	1	I, R, D	Hyperbolic tangent.

Statement Functions

Statement functions are function subprograms that can be defined in a single expression within the calling program. The definition is valid only in the program or subprogram containing it.

Form	Examples
$f[a_1, a_2, a_3, \ldots, a_n] = e$	NUMBER [K] =K*K(K+1)/2 EI [THETA] =COMPLEX [COS [THETA]] QTH [OM] =NAME [OM] +ADDR [OM] SWITCH CK [A, B, C] =FLAG [A] .AND. FLAG [B] .AND. FLAG [C]

where f is the function identifier, the a_i are dummy function arguments[†], and e is an expression.

Once a statement function has been defined, the appearance of its name in an expression is sufficient to call the function during the evaluation of the expression at run time. The function name is accompanied by the actual arguments to be used in evaluating the expression. For example, if the function were defined as

F X =A*X**2+B*X+C

it might be referenced in the program statement

RESULT = F [Y]

The current value of Y would replace the dummy argument X in evaluation of the function. The value of the expression

A*Y**2+B*Y+C

would be returned to the calling program where it would be assigned to RESULT.

A statement function must have at least one argument. The expression e must be of a mode that may be assigned to data of the type declared (implicitly or explicitly) for the function f. References in the expression are unrestricted with the exception that the identifier of the function f itself may not appear; however, any other statement function may be referenced.

Since each a_i is merely a dummy and as such does not actually exist in storage, the identifiers used to represent the a_i may be the same as any other identifier, except those referenced within the expression e, without conflict.

If it is implicitly typed, a statement function is considered integer type if its identifier begins with I, J, K, L, M, or N; otherwise it is considered real type. If a statement function is to be typed explicitly, its identifier must appear in a Type statement prior to the definition of the function.

As stated previously, a statement function may be referenced only within the program or subprogram in which it is defined. Statement function definitions must precede all executable statements in the program or subprogram in which they appear.

FUNCTION Subprograms

Functions that cannot be defined in a single statement may be defined external to the calling program as FUNCTION subprograms. Like statement functions, the FUNCTION subprogram computes a value and returns that value through the function identifier.

The FUNCTION subprogram must begin with a function statement.

[†]Dummy arguments serve as placeholders for the actual arguments provided by the calling program at execution time. Since the rules governing dummies apply to all subprograms, they are discussed separately under "Dummy Arguments".

Form	Examples
FUNCTION f [$a_1, a_2, a_3, \ldots, a_n$]	FUNCTION DIFFEQ [R, S, N] FUNCTION IOU [W, X, Y, Z1, Z2] FUNCTION ROUND [OMEGA]

where f is the function identifier, and the a_i are dummy arguments.

Each FUNCTION subprogram must have at least one argument. Values may be assigned to arguments within the subprogram without restrictions.

A FUNCTION subprogram must contain at least one RETURN statement to transfer control back to the calling program. A return is the last statement executed in the function (see Chapter 7).

Within the subprogram, the identifier of the FUNCTION is treated as though it were a scalar variable, and must be assigned a value during each execution of the subprogram. The value returned for a FUNCTION is the last one assigned to its identifier prior to the execution of a RETURN.

A FUNCTION subprogram typed implicitly is considered to be integer type if its identifier begins with the letters I, J, K, L, M, or N; otherwise it is considered to be real.

An example of a FUNCTION subprogram to find the product of two 1-dimensional arrays with 3 elements each:

```
      FUNCTION DOT [V1, V2]
      DOT=0
      DO 2 K=1, 3
2     DOT= DOT + V1 (K) * V2(K)
      RETURN
      END
```

If this function is called by the statement

```
PROD=DOT [A, B]
```

The dummies V1 and V2 will be replaced by A and B at execution time. The value of the function is the single quantity DOT, which will be returned to the calling program and assigned to the variable PROD. Note that while A and B must be dimensioned in the calling program, V1 and V2 should not be dimensioned in the function (see "Dummy Arguments").

SUBROUTINE SUBPROGRAMS

SUBROUTINE subprograms, like function subprograms, are self-contained programmed procedures. However, unlike functions, SUBROUTINEs do not have values associated with their identifiers, and may not be referenced in an expression. Instead, SUBROUTINEs return values to the calling program by assigning values to arguments, and are accessed by CALL statements (see Chapter 7).

Subroutine subprograms always begin with a SUBROUTINE statement.

Form	Examples
SUBROUTINE p [$a_1, a_2, a_3, \ldots, a_n$]	SUBROUTINE CHECK SUBROUTINE VII [ROMAN] SUBROUTINE OUTPUT ARRAYS [A, FMT, I, J]

where p is the SUBROUTINE identifier, and the a_i are SUBROUTINE dummy arguments. (See "Dummy Arguments".)

If no arguments are to be passed to the subroutine by the calling program or subprogram, the list of a_i and the comma and bracket characters would not be present; otherwise they are required.

A subroutine must contain at least one RETURN statement to transfer control back to the calling program; it must be the last statement to be executed during a run.

The following is an example of a SUBROUTINE subprogram which finds the cross product of two 1-dimensional arrays with 3 elements each:

```
SUBROUTINE CROSS [A, B, C]
C(1) = A(2) * B(3) - A(3) * B(2)
C(2) = A(1) * B(1) - A(1) * B(3)
C(3) = A(1) * B(2) - A(2) * B(1)
RETURN
END
```

The call to this subroutine might be:

```
CALL CROSS [X, Y, Z]
```

in which case the current values for array elements in X and Y would be used during execution of the subprogram, and the results would be assigned to elements in Z. Note that although X, Y, and Z should be dimensioned in the calling program, A, B, and C should not be dimensioned in the subroutine (see "Dummy Arguments").

Since the name of a SUBROUTINE plays no part in the result, it has no bearing on the mode of the result.

DUMMY ARGUMENTS

Dummy arguments provide a means of passing information between a subprogram and the program that called it. Both Function and SUBROUTINE subprograms may have dummy arguments, but a SUBROUTINE need not have any, while a Function must have at least one. Dummies are merely "formal" arguments, used to indicate the number and sequence of subprogram parameters. They do not actually exist in that no storage areas are used for them.[t] They serve as placeholders for the calling arguments.

The calling arguments may be scalar variables, array elements, array names, expressions, or subprogram identifiers. However, the dummies corresponding to these are always written as unsubscripted identifiers.

A dummy argument need not conform to the data type of the corresponding calling argument, should not be defined in a Type statement within the subprogram, and should not be dimensioned in the subprogram. In other words, declarations in effect for calling arguments at execution time are those which prevail during execution of the subprogram.

When a dummy corresponds to a variable in the calling argument list, a reference to the dummy is actually a reference to the calling argument variable. Not only will the dummy initially have the value to which the calling argument was assigned at the time of the call, but any value subsequently assigned to the dummy will actually be assigned to the calling variable, thus effectively returning a result through the argument list. For example; if the calling statement for a function subprogram is

```
Y=X**NI+SQRTDS[Z, Q]
```

and the function called is

```
FUNCTION SQRTDS[A, B]
C=AMAX1[A, B]
B=AMIN1[A, B]
```

---

[t] More precisely, storage areas are associated with dummies but contain pointers back to the storage area for the calling arguments.

```
A=C
SQRTDS=SQRT [A**2-B**2]
RETURN
END
```

then the values of Z and Q will be reversed whenever the initial value of Q is greater than that of Z.

When a dummy corresponds to an expression other than a single variable, the expression serves to initialize the value of the dummy. In this case storage is actually reserved for the dummy, whose value may be modified within the subprogram. For example, if the constant 3 in the calling list corresponds to the dummy J in the subprogram list, J will be initialized to 3 and may be modified in the subprogram. However, no result is returned through J and the value of the constant in the calling program is not affected.

APPENDIX. EXECUTION DIAGNOSTIC MESSAGES

If an error occurs during execution, the program is terminated, and the statement in error and one of the following messages is printed on the teletype listing.

ERRORS IN LIBRARY ROUTINES

1. ARGUMENT IN COMPLEX LIBRARY FUNCTION IS NOT REAL, INTEGER OR COMPLEX.
2. ARGUMENT IN EXTENDED PRECISION LIBRARY FUNCTION IS NOT REAL, INTEGER OR EXTENDED PRECISION.
3. ARGUMENT IN REAL LIBRARY FUNCTION IS NOT REAL, INTEGER OR EXTENDED PRECISION.
4. ERROR IN COMPLEX ARCTANGENT.
5. ERROR IN HYPERBOLIC FUNCTION.
6. EXPONENT TOO LARGE IN EXPF FUNCTION.
7. MORE THAN ONE ARGUMENT IN A LIBRARY FUNCTION REQUIRING ONLY ONE.
8. NEGATIVE ARGUMENT IN SQUARE ROOT.
9. NEGATIVE OR ZERO LOGARITHM.
10. NOT TWO ARGUMENTS IN A TWO ARGUMENT LIBRARY FUNCTION.
11. NUMBER TOO LARGE TO BE INTEGERIZED.

ERRORS IN I/O AND DATA STATEMENTS

1. A FORMATTED OPERATION TO A BINARY FILE OR A BINARY OPERATION TO A SYMBOLIC FILE.
2. ALPHABETIC DATA CAN ONLY BE STORED IN A REAL VARIABLE.
3. AN ATTEMPT HAS BEEN MADE TO OUTPUT A VARIABLE WHICH HAS NEVER BEEN STORED INTO.
4. BOOLEAN VARIABLES CAN BE OUTPUT ONLY WITH AN "A" FORMAT.
5. FIELD WIDTH GREATER THAN 63.
6. FIELD WIDTH LESS THAN NUMBER OF DIGITS AFTER DECIMAL POINT.
7. FILE CAN NOT BE OPENED - MAY ALREADY BE OPEN.
8. FILE DESIGNATOR IS NOT IN RANGE 0-4.
9. FILE HAS NEVER BEEN OPENED.
10. FILE NAME NOT IN DIRECTORY.
11. FORMAT CONTAINS A STRING HAVING AN ILLEGAL PRIME QUOTE COUNT.
12. FORMAT PART EXCEEDS I/O BUFFER SIZE.
13. FORMAT NOT A STRING VARIABLE.
14. ILLEGAL CHARACTER IN FORMAT.
15. ILLEGAL CHARACTER IN INPUT FIELD.
16. ILLEGAL FORM OF SCALE FACTOR.
17. ILLEGAL HOLLERITH COUNT IN DATA STATEMENT.
18. ILLEGAL REPETITION NUMBER.
19. ILLEGAL RIGHT PARENTHESIS IN DATA STATEMENT.
20. ILLEGAL RIGHT PARENTHESIS IN FORMAT NEST.
21. INPUT FORM BINARY FILE TO A STRING OR TEXT VARIABLE NOT ALLOWED.
22. INPUT NUMBER IS TOO LARGE OR TOO SMALL.

23. INPUT STRING HAS AN ILLEGAL PRIME-QUOTE COUNT.
24. LOGICAL VARIABLE BEING INPUT WAS ALL BLANK.
25. LOGICAL VARIABLE DOES NOT START WITH T OR F.
26. NUMBER OF DIGITS AFTER DECIMAL POINT GREATER THAN 63.
27. ONLY AN INTEGER CAN BE USED FOR A SCALE FACTOR.
28. ONLY THE CHARACTER "A" IS ALLOWED IN "A" FORMAT PART.
29. ONLY THE CHARACTER "O" IS ALLOWED IN "O" FORMAT PART.
30. OUTPUT OF A STRING OR TEXT VARIABLE TO A BINARY FILE NOT ALLOWED.
31. OVERFLOW OF DATA STATEMENT NESTING STACK.
32. OVERFLOW OF FORMAT NESTING STACK.
33. READ FROM A FILE OPENED FOR OUTPUT OR OUTPUT TO A FILE OPENED FOR INPUT.
34. SCALE FACTOR TOO LARGE.
35. STRING VARIABLES CAN BE OUTPUT ONLY WITH AN "A" FORMAT OR WITH FREE FORM OUTPUT.
36. TERMINAL DOLLAR SIGN IS MISSING FROM TEXT IN FORMAT.
37. UNEXPECTED END-OF-FILE.
38. ZERO COUNT FOR HOLLERITH FORMAT PART.

MISCELLANEOUS ERRORS

1. A=0 AND B NON-POSITIVE IN A**B.
2. A IN A**B NOT TYPED REAL OR INTEGER.
3. A IS NEGATIVE AND B IS TYPED REAL IN A**B.
4. ATTEMPT TO DIVIDE BY ZERO.
5. ATTEMPTED ARITHMETIC WITH LOGICAL VALUE.
6. B IN A**B NOT TYPED REAL OR INTEGER.
7. CALL TO UNDEFINED EXTERNAL ROUTINE.
8. COMMON ILLEGALLY LENGTHENED BY "/'
9. DIMENSIONED VARIABLE HAS NO SUBSCRIPT.
10. DO INCREMENT APPEARS IN A DECLARATIVE STATEMENT.
11. DO MAXIMUM APPEARS IN A DECLARATIVE STATEMENT.
12. DO MINIMUM APPEARS IN A DECLARATIVE STATEMENT.
13. DO VARIABLE APPEARS IN A DECLARATIVE STATEMENT.
14. EXTERNAL ROUTINE NAME USED AS VARIABLE.
15. GO TO VARIABLE HAS NOT BEEN ASSIGNED TO A STATEMENT LABEL.
16. LABEL MISSING.
17. LOGICAL OPERATOR REQUIRES REAL OR INTEGER OPERANDS.
18. MULTIPLE DECLARATION OF "/'
19. NUMBER OF SUBSCRIPTS INCORRECT.
20. STORAGE CAPACITY EXCEEDED BY "/'
21. STORING LOGICAL VALUE INTO NON-LOGICAL VARIABLE.
22. STORING NON-LOGICAL VALUE INTO LOGICAL VARIABLE.
23. STRING AND TEXT FUNCTIONS NOT IMPLEMENTED.
24. SUBSCRIPT NOT TYPED REAL OR INTEGER.

25. SUBSCRIPT VALUE TOO LARGE.
26. SUBSCRIPT VALUE TOO SMALL.
27. SUBSCRIPTED VARIABLE NOT DIMENSIONED.
28. VALUE OF GO TO VARIABLE TOO LARGE.
29. VARIABLE HAS NOT BEEN ASSIGNED A VALUE OR STATEMENT.
30. $ LABEL IS MISSING.

INDEX

A

A format, 43
ACCEPT statement, 34
account number, 2
ALT MODE key, 1
arguments to subprograms, 64, 65
arithmetic
  expressions, 20
  IF statement, 29
  operators, 20, 21
array
  declarations, 53
  elements, 19, 55
  identifiers, 19, 52
  storage, 54
  subscripts, 19, 54
  variables, 19
arrow
  backward (←), 13
  upward (↑), 12, 20
ASCII, 16
ASSIGN statement, 27
Assigned GO TO statement, 28
assignment statements, 25
  label, (see ASSIGN statement)
  replacement, 25
asterisk (*)
  double, operator (exponentiation), 20, 21
  in column 1, 4
  operator (multiplication), 20

B

BASIC, 3
blank (b̸)
  in FORMATs, 51
  in input strings, 49
  in statements, 4

C

C in column 1, 4
CAL, 3
CALL statement, 31, 52, 58
character
  non-printing control, 1
  set, 3
  strings, 16, 35, 43, 45
classification
  of data types, 16
  of identifiers, 52
CLOSE statement, 36
colon (:), 4
command language, 2, 6
comments, 4
COMMON statement, 4, 55
COMPLEX
  data, 16
  statement, 58
  type declaration, 58
Computed GO TO, 28
constants, 16-18
continuation line, 3
CONTINUE command, 3
CONTINUE statement, 30
Control (CTRL) key, 1
conversion
  format, (see format specifications)
  in assignment statements, 26
  input/output, (see format specifications)
control statements, 27-32
  CALL, 31
  CONTINUE, 30
  DO, 29
  END, 32
  GO TO, 27, 28
  IF, 29
  PAUSE, 31
  RETURN, 32
  STOP, 31
  COPY command, 5, 6, 8

D

D format, 40
DATA statement, 56
dash (-), 2
data types
  complex, 16
  double precision, 16
  Hollerith, 16
  integer, 16
  logical, 16
  real, 16
declaration statements, 52-58
  array, 53
  COMMON, 55
  DATA, 56
  DIMENSION, 55
  explicit, 53
  implicit, 18, 52
DEFINITIONS command, 6, 9
DELETE command, 5, 6, 8
diagnostics, 12, 66-68
digits, 3
DIMENSION statement, 4, 55
dimensions of arrays, 53
DISPLAY statements, 35
DO statement, 29, 30
DO-implications
  in DATA statements, 56
  in I/O lists, 33
dollar sign ($) format, 44
DO loops, 30

DOUBLE PRECISION
  data, 16
  statement, 58
  type declaration, 58
dummy arguments, 64, 65
dummy identifiers, 64

E

E format, 39
END statement, 4, 32
ENTER command, 5, 6
ESCAPE key, 1, 2, 3
evaluation hierarchy
  arithmetic, 21
  logical, 24
executable statements, 3
EXECUTE command, 6, 10
executive system, 1, 2, 3
EXIT command, 3
explicit declarations, 53
exponentiation, 20, 21
expressions, 20-25
  arithmetic, 20
  evaluation hierarchy, 21, 24
  logical, 23
  mixed, 22
  relational, 22

F

F format, 38
FALSE, 18, 42, 43
field specifications, 37
field termination, 49
file directory, 11, 36
files, 5, 6, 10, 11, 36, 37
fixed-point data, (see Integer data)
floating-point data, (see Real data and Double-precision data)
FORMAT statement, 37
FORMAT and list interfacing, 50
format specifications (input/output),
  A, 43
  D, 40, 49
  dollar sign ($), 44
  E, 39, 49
  F, 38, 49
  G, 40, 49
  H, 44
  I, 41, 49
  J, 40, 49
  L, 42
  P, 46
  parenthesized, 48, 51
  quote mark ('), 45
  slash (/), 47
  X, 45
  Z, 48
FORMAT statement, 37
FORMATs stored in arrays, 51
formatted input/output, 35
FORTRAN program, 3
FUNCTION statement, 63
function references, 20, 58
functions, 20
  intrinsic, 58
  library, 58-61
  statement, 62
FUNCTION subprograms, 62

G

G format, 40
GO TO statements, 27, 28
  assigned, 28
  computed, 28
  unconditional, 27

H

H
  format, 44
  in Hollerith constants, 18
hierarchy, (see Evaluation hierarchy)
hollerith constants, 18, 43-45

I

I format, 41
identifiers, 18
  classification, 52
  command, 6
IF statements, 29
  arithmetic, 29
  logical, 29
IJKLMN rule of typing, 18, 20, 53
implicit declarations, 52
implicit type, 52, 53, 62, 63
implied DO loops, (see DO-implications)
incremental compilation, 1
initialization of variables, 56
input/output conversion (see format specifications)
input/output lists, 33
  DO-implied lists, 33
  simple lists, 33
input/output statements, 33-52
  FORMAT, 37
  formatted, 35
  free format, 34
  READ, 36
  WRITE, 37
INTEGER
  data, 16
  statement, 58
  type declaration, 58

J

J format, 40

L

L format, 42
label assignment statements, (see ASSIGN statement)
labels, 4, 27
letters, 3, 18
library subprograms, 58-61
limits on data values, 16
LINE FEED key, 1, 3
LIST command, 6, 8
lists
  Do-implied, 33
  simple, 33
LOAD command, 6, 11
local mode, 2
LOGICAL
  data, 16, 42
  expressions, 23
  IF statement, 29
  operators, 24
  statement, 58
log-in procedure, 2
LOGOUT command, 3
log-out procedure, 3

M

mixed expressions, 22

N

names, (see Identifiers)
nested DO loops, 30
nested repetitions in FORMATs, 48, 51
nonexecutable statements, 3
nonprinting control characters, 1
numeric input strings, 49

O

O format, 41
on-line, 3, 5
OPEN statement, 36
operators
  arithmetic, 20, 21
  logical, 24
  relational, 23
Originate (ORIG) key, 2
output lists, (see Input/Output lists)

P

P specification (scale factor), 46
parenthesized format specifications, 48, 51
password, 2
PAUSE statement, 31
pound sign (#), 4
precedence of operations, (see Evaluation hierarchy)
precision of data, 16
program file, 3, 5, 6, 10, 11
project code, 2

Q

QED, 3, 11, 12
quotation mark (') format, 45
question mark (?), 6

R

range
  of a DO, 29, 30
  of statements, 4, 5
READ
  formatted, 36
  statement, 36
REAL
  data, 16
  type declaration, 58
REFERENCES command, 6, 9
references to array elements, 55
relational expressions, 22
relational operators, 23
replacement statement, 25
RESEQUENCE command, 6, 9
RETURN key, 1, 3
RETURN statement, 32

S

sample program, 12-15
SAVE command, 6, 10
scalar variables, 19
scale factor (P specification), 46
semicolon (;), 3
slash (/)
  FORMAT specification, 47
  in DATA statement, 56
  in file name, 10
  operator (division), 20, 21
  specification, 47
special characters, 3
statement functions, 62
statement labels, (see Labels)
statement numbers, 4
statements, 4
  executable, 3
  nonexecutable, 3
  termination of, 4
STOP statement, 31
storage allocation declarations
  COMMON statement, 4, 55
subexpressions, 21-25
subprogram control, 31
subprogram definitions, 58
subprogram identifiers, 58
subprograms, 58-65
SUBROUTINE statement, 63
subroutine subprograms, 63
subscripts, 19

T

termination of input strings, 49
.TRUE., 18, 42, 43
truncation, 26, 34
type declarations, 57
type statement, 4, 57
  COMPLEX, 58
  DOUBLE PRECISION, 58
  INTEGER, 58
  LOGICAL, 58
  REAL, 58
types of data, (see Data types)
typographic conventions, 1

U

Unconditional GO TO statement, 27

V

variables, 16, 18
variables (cont.)
  array, 19
  scalar, 19

W

WRITE
  formatted, 37
  statement, 37

X

X format specification, 45

Z

Z specification, 48
zero
  tests for, 29

SDS 940 FORTRAN IV REFERENCE MANUAL

SCIENTIFIC DATA SYSTEMS • 1649 Seventeenth Street • Santa Monica, California 90404

EXECUTIVE OFFICES
1649 Seventeenth Street
Santa Monica, Calif. 90404
(213) 871-0960

DEVELOPMENT DIVISION
2525 Military Avenue
Los Angeles, Calif. 90064
(213) 879-1211

MANUFACTURING DIVISION
555 South Aviation Blvd.
El Segundo, Calif. 90245
(213) 772-4511

MARKETING DIVISION
1649 Seventeenth Street
Santa Monica, Calif. 90404
(213) 871-0960

SYSTEMS DIVISION
555 South Aviation Blvd.
El Segundo, Calif. 90245
(213) 772-4511

600 East Bonita Avenue
Pomona, Calif. 91767
(714) 624-7371

12150 Parklawn Drive
Rockville, Maryland 20852
(301) 933-5900

PROGRAMMING
2526 Broadway Avenue
Santa Monica, Calif. 90404
(213) 870-5862

INSTRUMENTS
555 South Aviation Blvd.
El Segundo, Calif. 90245
(213) 772-4511

TRAINING
1601 Olympic Boulevard
Santa Monica, Calif. 90404
(213) 871-0960

SALES OFFICES

EASTERN

Maryland Engineering Center
12150 Parklawn Drive
Rockville, Maryland 20852
(301) 933-5900

69 Hickory Drive
Waltham, Mass. 02154
(617) 899-4700

1301 Avenue of the Americas
New York City, N. Y. 10019
(212) 765-1230

12 Huntington Brook
Rochester, New York 14625
(716) 586-1500

One Bala Avenue Building
Bala Cynwyd, Pa. 19004
(215) 667-4944

SOUTHERN

Holiday Office Center
Suite 4
3322 South Memorial Pkwy.
Huntsville, Alabama 35801
(205) 881-5746

Washington Plaza North
Suite 111
3880 Highway U.S. 1 South
Titusville, Florida 32780
(305) 267-6181

2964 Peachtree Road, N.W.
Suite 350
Atlanta, Georgia 30305
(404) 261-5323

8383 Stemmons Freeway
Suite 233
Dallas, Texas 75247
(214) 637-4340

3411 Richmond Avenue
Suite 202
Houston, Texas 77027
(713) 621-0220

MIDWEST

2720 Des Plaines Avenue
Des Plaines, Illinois 60018
(312) 824-8147

17500 W. Eight Mile Road
Southfield, Michigan 48076
(313) 353-7360

Suite 222, Kimberly Bldg.
2510 South Brentwood Blvd.
Brentwood, Missouri 63144
(314) 968-0250

One Parkway Center
Pittsburgh, Pa. 15220
(412) 921-3640

WESTERN

1360 So. Anaheim Blvd.
Anaheim, Calif. 92805
(213) 865-5293

2526 Broadway Avenue
Santa Monica, Calif. 90404
(213) 870-5862

505 W. Olive Avenue
Suite 300
Sunnyvale, Calif. 94086
(408) 736-9193

World Savings Bldg.
Suite 401
1111 So. Colorado Blvd.
Denver, Colo. 80222
(303) 756-3683

Fountain Professional Bldg.
9000 Menaul Blvd., N.E.
Albuquerque, N. M. 87112
(505) 298-7683

Dravo Bldg., Suite 501
225 108th Street, N.E.
Bellevue, Wash. 98004
(206) 454-3991

CANADA

864 Lady Ellen Place
Ottawa 3, Ontario
(613) 722-8387

INTERNATIONAL REPRESENTATIVES

FRANCE

Compagnie Internationale pour l'Informatique

EXECUTIVE OFFICES
101 Boulevard Murat
Paris 16eme

SALES OFFICES
17 Rue de la Reine
Boulogne 92

MANUFACTURING AND ENGINEERING
Rue Jean Jaures
Les Clayes Sous Bois 78

ISRAEL

Elbit Computers Ltd.
Subsidiary of Elron
Electronic Industries Ltd.
88 Hagiborim Street
Haifa

JAPAN

F. Kanematsu & Co. Inc.
Central P. O. Box 141
New Kaijo Building
Marunouchi, Chiyoda-Ku
Tokyo

90 11 15A