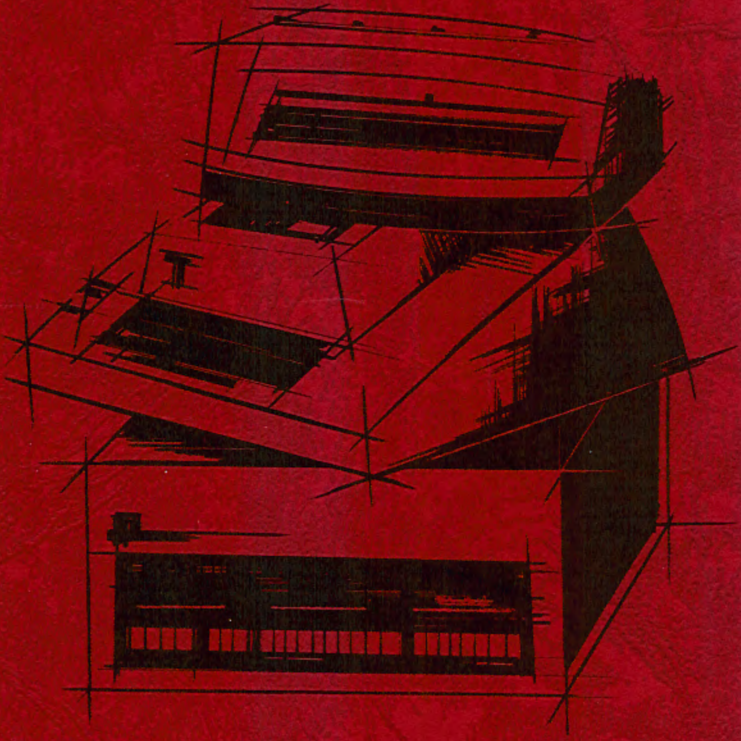
3300 BASIC REFERENCE MANUAL



WANG

3300 BASIC REFERENCE MANUAL

WANG

PREFACE

This manual covers the BASIC language specification for Wang 3300 Standard Basic and Wang 3300 Extended Basic, and terminal operating procedures of these WANG 3300 BASIC systems for use with the WANG 3310/11 I/O Typewriter terminal, the Wang 3315 Teletype* terminal and the WANG 3320 cassette tape drives.

Extended BASIC is generally identical to Standard BASIC except that it has additional features such as matrix operations, string variables, file operations, and expanded printing capability.

The operation and language of the WANG BASIC systems using the two types of terminals are quite similar. The main differences are found in the use of certain special characters in the language and in the terminal operating procedures.

If the reader has no fundamental knowledge of the BASIC language, it is recommended that the WANG 3300 BASIC BEGINNER'S PROGRAMMING GUIDE be read prior to the use of this manual.

*Trademark of the Teletype Corp., Skokie, Illinois

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

INTRODUCTION

The WANG 3300 BASIC system is a multi-terminal time-sharing system utilizing either the WANG 3300 Standard BASIC language or the WANG 3300 Extended BASIC language. These languages are an extension of the BASIC language which was originally developed at Dartmouth College and has obtained universal popularity. The extensions of the Dartmouth BASIC language found in WANG 3300 BASIC occur primarily in selective format changes including a one line program calculator mode and terminal operating commands. The languages are generally quite similar.

WANG 3300 Standard BASIC is essentially a compatible subset of WANG 3300 Extended BASIC. WANG Extended BASIC contains a number of additional features including matrix operations, alphanumeric string variables, file operations, and expanded printing capability. It requires a slightly larger amount of computer memory to hold the system.

The WANG 3300 BASIC systems consists of a WANG 3300 computer, WANG 3310 Typewriter terminals and WANG 3320 cassette tape drives or WANG 3315 Teletype terminals, or a combination of these. This manual contains the BASIC language specifications for both Standard and Extended BASIC and terminal operating procedures for both the 3315 Teletype and the 3310 Typewriter/3320 Cassette Tape terminal configurations. With these configurations a number of terminals and possibly cassette tape drives are located in the immediate vicinity of the computer. At the terminals, a number of users can simultaneously enter and execute programs in the BASIC language to solve a great variety of problems. The terminals are all linked to the WANG 3300 computer which communicates with the terminals and executes entered problems on a time-shared basis. Each user operates his terminal much like an electric typewriter. The BASIC language in which the problems are entered is relatively simple to learn. The use of it along with uncomplicated terminal operating procedures leaves the user free to concentrate on the problem being solved.

The first section of this manual, WANG 3300 BASIC, provides a general introduction to both the WANG Standard and Extended BASIC language and terminal procedures. The remaining sections deal with the definition and specifications of the WANG 3300 Standard BASIC language format, the additions to it for Extended Basic, system terminal commands, BASIC program statements, terminal operating procedures, error messages, and program limits. The appendices contain concise language syntax rules and programming examples.

WANG 3300 BASIC

WANG 3300 BASIC is a multi-user, conversational, time-sharing system for solving the problems in science, engineering, mathematics, and business. A user of the WANG system can program and run his application as one of 16 simultaneous users of the system.

Programs are written in the easy-to-learn BASIC language. This language consists of a combination of

English words: `LET, PRINT, READ`

Mathematical functions: `LOG, SIN, EXP`

Expressions: `Y + 2, (A + B)/C`

Variables: `C2, D, E`

A BASIC word and verb syntax form a BASIC statement. A series of statements form a program for the definition and solution of a problem. After a program has been entered, the user may then enter a BASIC command for storing his program on tape. Programs originated by the user may also be supplemented by a large number of interesting and practical library programs.

The standard terminals for system-user communication are the WANG 3315 Teletype and the WANG 3310 I/O Writer, which may be taken off-line and used as a conventional electric typewriter.

Communicating with BASIC

The preliminary steps for entering a program for solution by the BASIC system involve the WANG terminal. The turning on of the terminal and the striking of the ATTENTION key on the 3310 I/O Writer or the ESCAPE key (ESC) on the 3315 Teletype will cause the system to type out 3300 BASIC READY and a colon, thus:

```
 3300 BASIC READY
:
```

The words '3300 BASIC READY' mean that the system is prepared to receive instructions from the user; the colon indicates that the terminal now belongs to the user. He may type a command or proceed with entering his program. The terminal belongs to the user until he strikes a carriage return, which returns the terminal to the system. The colon also permits the user to identify the source of any line of the typeout -- lines preceded by a colon are user inputs, those without are system outputs. For example, a line may be entered for immediate execution:

```
: PRINT LOG (5.2 + 6↑3)
```

The system responds with the solution and a system input request -- the colon:

```
 5.3990673
:
```

The WANG BASIC system provides correction procedures that ease the initial use of the system and expedite its effectiveness.

If the user detects an error in a program text line he is currently entering, he may type a degree character (which is an uppercase E) on the 3310 I/O Writer or a blackslash character ( ) on the 3315 Teletype. This character will cause that line to be ignored. When the '°' or '\' is encountered, the system will remove the contents of the current line. The user may then reenter the correct line. For example,

```
      :120  LET X=AB/C°    on the 3310 I/O Writer
or
      :120  LET X=AB/C\    on the 3315 Teletype
```

The user may correct single characters in a line by using the backspace key on the 3310 I/O Writer or the backarrow (←) on the 3315 Teletype. This action 'crosses out' the most recently typed characters. Several characters may be so crossed out in succession, as shown in the following examples:

```
3315         :120  LET X=10+ ←←←20+J
Teletype       Three backarrows ←┘  ↓
               to remove last 3    user
               characters          correction
             :120  LET X=20+J  →  The line as it will be
                                  processed by the system
```

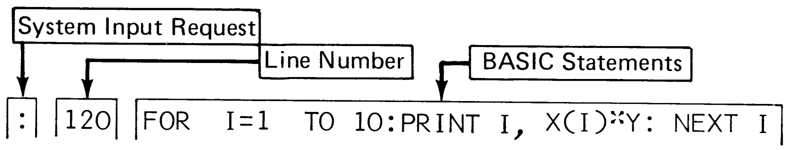
On the 3310 I/O Writer, the backspace sequence is completed by manually turning the carriage to advance the paper to the next line; the corrected text may then be entered.

```
                          ←←←
3310 I/O     :120  LET X=10+  → Three backspaces to
Writer                          remove last 3 chars.
                         20+J → User correction
             :120  LET X=20+J → Line as it will be processed
                                by the system
```

BASIC Program Mode

A BASIC program is a group of lines containing numbered executable and nonexecutable program statements. System commands direct program execution but are not part of the program itself. BASIC program statements are saved as they are entered for ultimate execution; commands, on the other hand, cause action as soon as they are entered and, therefore, are not saved.

A BASIC program line is one or more statements for processing or solution by the system. Each program line is comprised of a line number and at least one statement with verb syntax. A series of statements, separated by colons, may be entered on one line — with one line number. For example,



```
: 120 FOR  I=1  TO 10:PRINT I, X(I)*Y: NEXT I
```

Multiple statements on one line provide for compression of a program into fewer lines and are of great use in the immediate execution mode, described on page 5. A single statement per program line can, however, be considered the standard way to program in BASIC because corrections can be made more easily and references to an exact statement can be more precise.

The numbered lines may be entered in any order; the system sorts them into ascending numeric order before execution. If two lines should have the same number, the line last entered replaces the first and is used by the system. A user may delete a line previously entered simply by entering the line number and striking the carriage return.

Standard BASIC program statements are typed one line at a time into the system until a complete BASIC program has been entered. After each line is received, the system performs a syntax error check, types an error message if errors are found, and saves the line. When all program lines have been entered and corrected, the BASIC program is executed by the entry of a command (RUN).

An example of a simple program for the solving of a quadratic equation and the commands for its execution is shown below:

The equation:

$$2x^2 + 9x + 3 = 0$$

The method of solution:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

for b = 9, a = 2, c = 3.

The BASIC program:

```
  3300 BASIC READY
: START → BASIC command to initialize the system
: 10 REM THIS IS PROBLEM 1 → A nonexecutable statement
: 20 LET B = 9: LET A = 2: LET C = 3 → BASIC statements
: 30 LET X = (-B + SQR(B↑2- (4*A*C)))/(2*A)
: 40 PRINT X
: RUN → BASIC command to execute the program
```

Immediate Execution Mode

BASIC program execution requires that each statement be preceded by a line number; this is 'program mode' entry. WANG BASIC provides an 'immediate mode' entry for the execution of a single program line. This mode provides, in effect, an implicit one line program and execution. It may also be used to set and/or examine values and expressions or perform calculations immediately after they are entered. The immediate mode entry follows the format rules of standard program entry with one exception: the line number is not entered at the beginning of the statement line. The missing line number tells the system that this line is to be executed immediately. Once entered and executed, immediate mode lines are not saved, as are standard program statements.

The following examples are valid immediate mode entries and demonstrate the usefulness of this mode:

```
PRINT (4.2* SIN(1.456-4*.033))-1.2E1
LET X = 2: LET Y = 3: PRINT X, Y + EXP(X*Y)
FOR I = 1 TO 6: PRINT I, X(I)*Y: NEXT I
```

The first example – the PRINT statement – allows the system to act as a conversational calculator. The system performs the calculations and immediately prints out the answer. The LET statement shows the use of the immediate mode for setting variable values and for displaying values computed from them; the FOR, TO statement demonstrates the use of one-line loops – here to display a function of a previously defined array (arrays are described on page 13).

Any BASIC verb (except INPUT and those which reference a line number, e.g., GO TO) may be used in the immediate mode; some verbs, however, have little meaning which will be obvious to the user when he is composing the program line. The verbs customarily used are:

```
FOR
LET
NEXT
PRINT
RESTORE
TRACE
```

BASIC Commands

A BASIC command is simply a request to the system for the performance of a particular function. For example, START initializes the system to receive a new program by removing any previous program text and variables: RUN executes the current program. Some commands are followed by a line or unit number: RERUN 60 means run the program beginning with line 60 without re-initializing the current values of the variables; SAVE 3 means record this program on tape number 3. Other commands have no statement or unit references: RUN, START, RESTART. System commands, unlike BASIC program statements, are not saved by the system.

The ATTENTION key on the 3310 I/O Writer or the ESCAPE (ESC) key on the 3315 Teletype may be used as a special 'command'. If, during program execution, the user wishes to interrupt the system processing, he strikes the ATTENTION or ESCAPE key. Program execution will be terminated and the terminal returned to the user with the following response:

```
 3300 BASIC READY
:
```

Program Start and Execution

A BASIC program is initialized by entry of a START or RESTART command. The START command initializes the entire user program area, removing all previously stored program text and variables. The RESTART command, on the other hand, initializes the user program area but does not remove common variable data to be used by successive programs. After entering one of these commands, the user may then proceed with the entering of his program.

A RUN or RERUN command requests execution of a user's program. The major difference between RUN and RERUN is that RUN causes the entire program to be run from the beginning and all variables to be reset to initial values. RERUN, on the other hand, does not reinitialize values (leaves them at the current setting). The RERUN command may also be used to run a portion of a program; e.g., RERUN 60 means to rerun the program beginning with program line 60.

Program execution is terminated when a STOP or END statement is encountered in the program or when the user depresses the ATTENTION or ESCAPE key on the terminal or when an error is detected.

BASIC Error Detection

One of the features of the WANG 3300 BASIC system is its almost instant error detection and reporting. The error reporting system can point out an error in the user text as it is entered.

When an error is detected, the text line being scanned by the system is typed (if it is not already on the typewriter sheet), and, on the next line, an up-arrow (↑) is placed at the point of the current scan, followed by the error message number. The example below shows an error detected for unbalanced parenthesis; the message number for this error is 05. The complete set of error codes is listed in a later section of this manual.

```
:240 X=SIN ((Y-P1)+5
                    ↑ERR 05
:
```

The colon on the line following the error-detection line gives the system back to the user. He may then correct the line or condition causing the error.

Sharing of Common Data

The sharing of data common to several programs is a significant feature of the WANG BASIC system. A COM statement is entered as the first numbered line in a program to allow the storage of information in memory for sharing by a subsequent program. There may be many COM statements in one program — just as long as they are the first numbered executable lines. The attributes assigned to COM statements must be the same when entered in subsequent programs, which must be initialized by a RESTART command. An example of a program using COM statements is shown on the following page.

```
:   START
:   1 REM THIS IS PROGRAM 1
:   2 COM A(15), B(6), C
:┐
:├────────────►Program statements
:┘
:   RUN
:   RESTART
:   1 REM THIS IS PROGRAM 2
:   2 COM A(15), B(6), C, D(10)
:   3 COM E(4,14), F(3), G(6)
:┐
:├────────────►Program statements
:┘
:   RUN
```

BASIC Program Storage

BASIC programs which will have frequent use by one or more programmers may be written on tape for future entry into the system. The SAVE command causes all user text presently in core to be punched on paper tape on the user teletype terminal or written on a specified cassette tape. The LOAD command is used to read and append the file of user text currently loaded on a teletype paper tape or written on a specified cassette tape to the end of the user text in core. This command permits either additions to user-typed text or entry of a complete, previously saved program.

BASIC PROGRAM STRUCTURE AND COMPONENTS

A BASIC program must have a certain 'structure' – simple though it is. The rules are few and easy to follow. The structure of the program implies the use of certain components. These components include the allowable characters, the kinds of symbols, and the various functions that BASIC can use.

The following pages contain explanations of these rules and requirements.

Line Number

For BASIC program execution, every program statement must be assigned a line number. Line numbers not only identify the lines but specify the order in which the program lines are to be executed by the system. These lines do not have to be entered in sequence order; the BASIC system automatically processes the lines in order by line number. Line numbers should be assigned so that there are unassigned numbers between consecutive statements for the insertion of additional lines.

The line number can be one to four digits in length.

BASIC Program Lines

A BASIC program line is one or more statements for processing or solution by the system. Each program line is comprised of a line number and at least one statement with verb syntax. A series of statements, separated by colons, may be entered on one line – with one line number. For example,

```
: 40 LET X = 2: LET Y = 3: PRINT X,Y
```

There are two types of statements: an executable statement specifies the action to be performed. For example,

```
LET Q = 8*Y
```

A nonexecutable statement provides information. For example,

```
DATA 2, -7,5
```

Spaces are customarily used between characters in a program line for readability; the system ignores them. For example, 10 READ, A, B, C, D is easier for the programmer to read than 10READA,B, C,D; both, however, are equally clear to the BASIC system. The condensed format conserves user text area space.

Program Line Format

The formats for standard BASIC program lines and for immediate execution lines are shown below. A line number is required for the standard program mode but not for the immediate execution mode.

```
Program Mode    : 100 LET X=(C*E)/G
Immediate Mode  : PRINT(C*E)/G
```

The colon (:) is typed by the system to indicate that the programmer may proceed to enter his lines. This symbol is also useful for identifying lines in the program typeout – those preceded by a colon were entered by the user; those without are system output.

Expressions

An expression may be a variable, a function, or a constant; it may also be a combination of variables and constants connected by arithmetic symbols. An expression may be preceded by plus or minus and may be contained within parentheses. The following examples demonstrate the variety as well as the simplicity or complexity of BASIC expressions.

$$\texttt{LET X = }\boxed{\texttt{A}}$$
$$\texttt{LET X = }\boxed{\texttt{5*Y+FNB(X) - LOG(Z)}}$$
$$\texttt{LET J(}\boxed{\texttt{X2+5}}\texttt{,K)=9}$$
$$\texttt{FOR I = }\boxed{\texttt{3+K2}}\texttt{ TO }\boxed{\texttt{4*Y}}\texttt{ STEP }\boxed{\texttt{D(}\boxed{\texttt{3+K}}\texttt{) - 1}}$$
$$\texttt{PRINT }\boxed{\texttt{SIN(K) - 4*J}}$$

These are all expressions.

Expressions are executed by the BASIC system in the order prescribed below; when the operation is not explicit, it is executed from left to right in the expression.

1. The formula within parentheses is executed before the parenthesized quantity can be used in additional executions in the expression and before any other single formulas outside the parentheses.
2. When there are no parentheses in the expression and the operations are to be executed on the same level, these operations are performed from left to right. For example, in the expression A * B/C, B is multiplied by A and the product is divided by C.

3. Operations in an expression are executed in sequence from highest level to lowest level, as follows:

   -Operations within parentheses
   -Exponentiation (↑)
   -Multiplication or division (* or /)
   -Addition or subtraction (+ or –)

BASIC Verb Syntax

The following editorial rules are used in this handbook to define and illustrate the components of BASIC program statements and system commands.

1. Uppercase letters (A through Z), digits (0 through 9). and special characters (*, /, +, etc.) must always be used for program entry exactly as they are shown in the format examples.
2. Information in lowercase letters is to be supplied by the user; for example, in the program line 'GOSUB line number,' the line number must be entered by the user.
3. Square brackets, [ ], indicate that the enclosed information is optional. For example,

   ```
   RESTORE    [integer]
   ```

   means that the RESTORE statement verb can be optionally followed by an integer,

   ```
      RESTORE
   or
      RESTORE 2
   ```

   are both legal forms.
4. Braces, { }, enclosing vertically stacked items indicate that one of the items is required. For example,

   $$\text{COM} \left\{ \begin{array}{c} \text{scalar variable} \\ \\ \text{array variable} \end{array} \right\} ---$$

   means that the COM statement elements can be:

   a scalar variable; e.g., C2

   or

   an array variable; e.g., D (4, 8)

5. Ellipsis, ... , indicates that the immediately preceding item may occur one or many times in succession. For example,

```
INPUT variable, variable, ...
```

6. Except within quotation marks, BASIC syntax ignores blanks.
7. When one or more items appear in sequence, these items or their replacements must appear in the specified order.

BASIC Characters

A BASIC program is entered by means of a typewriter terminal which provides the following characters:

1. Twenty-six uppercase alphabetic characters: A through Z.
2. Numeric digits: 1 2 3 4 5 6 7 8 9 0.
3. Special characters:

'	Single quote	!	Exclamation mark	
"	Double quote	,	Comma	
&	Ampersand	.	Period	
@	At	;	Semicolon	
+	Plus	:	Colon	
−	Minus	<	Less than	
*	Asterisk	=	Equal to	
↑	Up arrow	>	Greater than	
/	Right oblique	#	Pound Sign	
(	Left parenthesis		Blank	
)	Right parenthesis			
≤	Less than or equal to			3310 I/O Writer only
≥	Greater than or equal to			3310 I/O Writer only
≠	Not equal to			3310 I/O Writer only
°	Degree			3310 I/O Writer only
\	Left oblique (backslash)			3315 Teletype only
←	Back arrow			3315 Teletype only

Numeric Constants

A numeric constant may be positive or negative and may consist of as many as eight digits. Entered numbers with greater than eight digits will result in an 'illegal number format' error. The following are examples of numeric constants in BASIC:

4, −10, 1432443, −.7865, 24.4563

If the exponential notation, E, is used, the value of the constant is equal to the number to the left of the E multiplied by 10 to the power of the number to the right of the E. For example, 4.5E7 indicates that '4.5' is to be multiplied by 10^7.

The magnitude of a numeric constant must be less than 1E+63 and greater than or equal to 1E-65.

Variable Names

A variable name is a string of characters that represents a data item that is operated on according to certain executable verbs such as READ, INPUT, LET, NEXT, FOR. The value assigned by the verb in a program line will not change until a second program line is encountered with a new value for the variable.

There are two types of variables: scalar and array. A scalar variable is designated by a letter or a letter followed by a digit:

A,A4

An array variable is used to define the elements of an array. These variables are used when a single subscript or a double subscript might ordinarily be used:

$(a_1, a_2, a_3, \ldots)$ or $b_{i,j}$

An array variable consists of a single letter (A through Z) which is the array name, followed by subscripts in parentheses. A(3), B(2,3) would be array variables. For all array variables, the DIM statement is used with the array name and the numeric value subscripts to provide space for a complete array of one or two dimensions. The DIM statement must precede the first reference to the variable. For example:

```
20 DIM Q(25)
30 READ N
40 FOR I = 1 TO N
50 READ Q(I)
55 PRINT Q(I)
60 NEXT I
70 DATA 5
80 DATA 4, 5, 19, 37, 43
```

For cases where an array variable will be used as Common Data, it is specified in a COM statement to provide storage space instead of a DIM statement.

The following rules apply to the use and assignment of array variables:

1. The numeric value of the subscript for the first array element must be 1; zero is not allowed.
2. The dimension(s) of an array cannot exceed 255.

Mathematical Functions

These functions have special names, followed by an expression in parentheses (or relation), as shown in the following examples:

SIN(3)	Find the sine of 3
COS(X + Y)	Find the cosine of the value of the expression X + Y

The following table provides the list of these functions and the BASIC interpretation of them.

Function	BASIC Interpretation
SIN(expression)	Find the sine of the expression (radians)
COS(expression)	Find the cosine of the expression (radians)
TAN(expression)	Find the tangent of the expression (radians)
ATN(expression)	Find the arctangent of the expression (radians)
EXP(expression)	Find e to the power of the expression
LOG(expression)	Find the natural logarithm of the expression
ABS(expression)	Find the absolute value of the expression
SQR(expression)	Find the square root of the expression
RND(expression)	Produce a random number between 0 and 1 (the expression is required but is meaningless here)
INT(expression)	Take the integral value of the expression
SGN(expression)	Assign the value 1 to any positive number, 0 to zero, and −1 to any negative number. For example: SGN(9.15) = 1 SGN(0) = 0 SGN(−.124) = −1
BOOL(relation)	Find the true value of the relation (result = 1 if relation true; = 0 if false).
AND(expression,.......)	Find the true value of the expressions (result = 1 if all expressions are non-zero; otherwise = 0).
OR(expression,.......)	(result = 1 if any expression is non-zero; otherwise = 0).

User Functions

A 'user function' defines a function which will be used several times within a program. Such a function is introduced by a DEF statement. The format of the function itself is FN followed by a letter or a digit, a scalar variable in parentheses; an equals sign, and an expression. A function could be used in a program as follows:

The function is defined:

```
30 DEF FNE(Z1) = EXP(-Z1↑3+5)
```

Then, if the following statement is entered,

```
40 LET Q = A/B + FNE(10)
```

the value of 10 will be assigned to Z1; the result,

```
EXP(-10↑3+5)
```

will be used in place of the referenced FNE(10) in program line 40.

Arithmetic Symbols

The following arithmetic symbols are used in BASIC to write a formula. Operations are executed in sequence from the highest level to the lowest level: (1) operations within parentheses (2) raising a number to the power, (3) multiplication and division, and (4) addition and subtraction.

Symbol	Sample Formula	Explanation
↑	A↑B	Raise A to the power of B.
⁘	A⁘B	Multiply B by A.
/	A/B	Divide A by B.
+	A+B	Add B to A.
–	A–B	Subtract B from A.

Relational Symbols

Relational symbols are used with the IF verb or the BOOL function when values will be compared before processing. For example,

```
20 IF G<10 THEN 63
```

which means that if G is less than 10, processing will continue at program line 63.

```
30 LET X = BOOL(A=B)
```

Assigns a value of one to X if A is equal to B, and zero if it is not.

Relational Symbols (continued)

The following relational symbols may be used with BASIC:

3315 Teletype Symbol	3310 I/O Writer Symbol	Sample Relation	Explanation
=	=	A=B	A is equal to B
<	<	A<B	A is less than B
<=	≤	A<=B, A≤B	A is less than or equal to B
>	>	A>B	A is greater than B
>=	≥	A>=B, A≥B	A is greater than or equal to B
<>	≠	A<>B, A≠B	A is not equal to B

BASIC SYSTEM COMMANDS

A BASIC command provides the user with a means for communicating with the system –– to request system operation. A command thus facilitates the running or modification of a program but is not part of the program itself. For example, the RUN command initiates the execution of an entered program; the SAVE command directs that all program text be punched on paper tape or written on a cassette tape.

BASIC commands are entered one line at a time. They differ from BASIC statements because they are not preceded with line numbers, they have unique verbs and only one command can be entered on one line; that is, multiple commands separated by colons on one line are not allowed. BASIC program statements are saved as entered for ultimate execution; BASIC commands cause action and are not saved.

The BASIC commands are described on the following pages.

LIST

Format

LIST [line number [, line number]]

Purpose

The LIST command will cause the typing out of user program text in statement number sequence; the extent of the listing depends upon the command format. If LIST is entered without a following line number, the complete program text is produced. If one line number is entered with the command, just that program line is produced. If two line numbers are entered, all text from the first through the second line numbers, inclusive, will be listed.

Example

```
LIST
30 READ A, B, C, M
.
.
.
990 END
```

or

```
LIST 30,50
30  READ A, B, C, M
40  LET G=A*D-B*C
50  IF G=0 THEN 60
```

or

```
LIST 30
30  READ A, B, C, M
```

LOAD (See SAVE also)

Format

LOAD unit

where unit is a tape cassette number and file, a disk file name, or blank.

Purpose

When the LOAD command is entered, the file of user text currently on the specified tape or on disk will be appended to the end of the user text in core. This command permits either additions to user-typed text or -- if entered after a START or RESTART command -- entry of a complete, previously saved program.

If no unit is specified, the system will assume the tape is to be loaded from the user teletype terminal paper tape reader and will automatically start the tape motion and read the tape.

If a tape number (a number from 1 to 32) is specified, it identifies a logical cassette tape from which the program will be loaded. The currently positioned file will be read. Optionally, if a / and a file number follow the tape number, the tape will be rewound and skip n—1 files before reading (where n is the specified file number.)

A disk file will be read if unit equals a disk file name. A disk file name is a string of up to 8 characters enclosed in quotes (i.e., "MASTER" or "FILE34").

Example

LOAD	(Load from Teletype)
LOAD 2	(Load from cassette number 2, current file)
LOAD 3/2	(Load from cassette number 3, second file)
LOAD "MASTER"	(Load from disk, file "MASTER")

RERUN (See RUN also)

Format

RERUN [line number]

Purpose

There are two uses of the RERUN command: if a line number is entered with the command, program execution will begin at the referenced line number. Otherwise, program execution will start at the lowest numbered program line.

The RERUN command causes the execution of the program to occur without reinitiating program variables to zero; the variables are maintained at the last calculated values. This differs from the RUN command which initiates all variables to zero.

The entire text area will be scanned by the system. If newly entered common (COM statements) variables are encountered, an error is assumed; such variables are, however, allowed for a RUN command. New noncommon variables are established with values of zero.

Example

RERUN $\longrightarrow$ This implies starting at the lowest program line number.

OR

RERUN 20 $\longrightarrow$ Rerun the program beginning with program line 20.

RESTART (See START)

Format

```
RESTART
```

Purpose

This system command will reinitiate the entire user program area, except that previously defined common variables (COM statements) are not disturbed. All user program text and noncommon variables are removed from the system, but names, attributes, and values of common variables are not changed.

Example

```
RESTART
```

RUN (See RERUN also)

Format	RUN
Purpose	The RUN command initiates the execution of the user's program. The system verifies all numbers and entries; variables are scanned for consistency; new (not previously entered) common variables and all noncommon variables are reset to zero. The program statements are then executed in line-number sequence.
Example	RUN

SAVE

Format

SAVE unit
where unit is a tape cassette number and file, a disk file name, or blank.

Purpose

This system command will cause all program lines to be written on disk or on the tape specified by the SAVE command. If the specified tape is not available, an error message is written and no action is taken.

If no unit is specified, the system will assume the text is to be punched on the user teletype terminal. Five inches of leader/trailer code will be punched before and after the program text. The user should manually turn the teletype paper tape punch on and off during the period when the leader/trailer code is being punched.

If a tape number (a number from 1 to 32) is specified, it identifies a logical cassette tape on which the program will be written (at the currently positioned file).

Optionally, if a / and a file number follow the tape number, the tape will be rewound and skip n—1 files before writing (where n is the file number). In this case, all subsequent files on the tape will be destroyed.

The program will be written on a disk file if unit equals a disk file name. Disk file names must be enclosed in quotes (i.e., "MASTER" or "FILE34"). If more than one disk storage unit (1024 characters) are required. The number of storage units needed must be specified by following the file name with a / and the number needed (i.e., "MASTER"/4 indicates that 4 storage units of 1024 characters are required).

Example

`SAVE`	(Punch on teletype)
`SAVE 4`	(Write on cassette number current file)
`SAVE 3/4`	(Write on cassette number 3, fourth file)
`SAVE "MASTER"/4`	(Write on disk, 4 storage units)

START (See RESTART also)

Format	START
Purpose	The START command will reinitiate the entire user program area. All previously stored program text and both common and non-common variables are removed from the system.
Example	`START`

BASIC STATEMENTS

A BASIC statement is a special verb or word followed by an expression, variables, or numbers. For example:

READ A, B $\longrightarrow$ A statement: verb followed by variables

DATA 1, 4 $\longrightarrow$ A statement: verb followed by numbers

LET A = 6*B $\longrightarrow$ A statement: verb followed by a variable (A), an equals sign, and an expression (6 * B).

There are two types of BASIC statements: executable and nonexecutable. An executable statement specified program action:

```
READ A, B
LET A = 6*B
```

A nonexecutable statement provides information for program execution:

```
DATA 1, 4
```

or for the programmer:

```
REM THIS IS PROGRAM 1
```

A series of statements, separated by colons, may be entered on one line – with one line number. For example:

```
20 FOR I = 1 TO 10: PRINT I,X(I)*Y: NEXT I
```

BASIC statement lines must always begin with a line number; statement lines for immediate execution do not.

COM

Format

$$\text{COM} \left\{ \begin{array}{l} \text{scalar variable} \\ \text{array variable} \end{array} \right\} \left[, \left\{ \begin{array}{l} \text{scalar variable} \\ \text{array variable} \end{array} \right\} \ldots \right]$$

Purpose

The COM statement allows a program to store information in memory for use in a subsequent program or to use information from a previous program. It provides array definition identical to the DIM statement for array variables; on the other hand, the syntax for one COM statement can be a combination of array variables (A(10), B(3,3)) and scalar variables (C2,D).

The common area variables must be defined before any other variable in the program is defined. Therefore, COM statements must be assigned the lowest executable line numbers in the program.

The following general rules apply to the COM statement:

1. It must be the first executable program line(s) in a program.
2. It must be used with identical attributes in a previous or subsequent programs.
3. Subsequent programs are initiated by a RESTART command.
4. It cannot be added to a program prior to a RERUN command.

Example

```
10 COM A(10), B(3,3), C2
20 COM C,D(4,14), E3, F(6)
```

DATA (See READ and RESTORE also)

Format

DATA n [{,n} ...]

where n is a number.

Purpose

The DATA statement provides the values to be used by the variables in a READ statement. The numbers entered with the DATA statement are in the order that they are to be used. If several DATA statements are entered, they are used in order of statement number.

Example

```
40 DATA 4, 3, 5, 6
50 DATA 6.56E + 45, -644.543
```

DEF

Format

DEF FNa(v) = expression

where a is a letter or digit which identifies the function and v is a scalar variable (a letter or a letter followed by a digit).

Purpose

The DEF statement defines a user's unique functions. The following program lines illustrate how DEF is used.

```
010 X = 3
020 DEF FNA(Z) = Z↑2 - Z
030 PRINT X +FNA(2*X)
040 END
```

Processing:

Evaluate the expression for the scalar variable. (i.e., 2*X)
Find the FN.
Set the scalar variable equal to the evaluated expression value. (i.e., Z=2*X)
Evaluate the FN expression. (Z↑2-Z)
The above example would print the value 33: 3+(6↑2-6).

The DEF statement may be entered anyplace in the program, and the expression may be any formula which can be entered on one line. A function cannot refer to itself; it can, on the other hand, refer to other functions. Up to five levels of function nesting are permitted. Two functions cannot refer to each other (an endless loop). A reference cannot be made to an outside DEF FN statement from an immediate execution mode statement. The scalar variable used in a DEF statement is called a dummy variable. It may have a variable name identical to a real variable used elsewhere in the program or in other DEF statements; current values of these variables will not be affected during FN evaluation.

Example

```
60 DEF FNA(C) = (3*A) - 8/C+FNB(2-A)
70 DEF FNB(A) = (3*A) - 9/C
80 DEF FN4(C) = FNB(C) *FNA(2)
```

DIM

Format

DIM array variable $\left[\left\{, \text{array variable}\right\} \ldots\right]$

Purpose

This statement reserves space for single or double-dimension array variables which will be referenced in the program. Space may be reserved for more than one variable with a single DIM program line by separating the entries for array name and integer with commas, as shown in the example below.

DIM statements must appear before any use of the variable in the program, and the space to be reserved must be explicitly indicated – expressions are not allowed.

The following rules apply to the use and assignment of array variables in a DIM statement.

1. The numeric value of the first subscript must be 1; zero is not allowed.
2. The dimension(s) of an array cannot exceed 255.

Example

```
20 DIM I(45) ———►Reserves space for a single-dimension array of 45
                 elements
30 DIM J(8,10) ———►Reserves space for a double-dimension array of 8
                   rows and 10 columns
40 DIM K(35), L(3), M(8,7) ———►Reserves space for two single-
                               dimension and one double-
                               dimension array
```

END

Format

END

Purpose

This is an optional statement: It indicates the end of a BASIC program. It need not be the last executable statement in a program. An 'END PROGRAM' is typed by the system when this statement is executed, and the system then awaits subsequent user action.

Example

```
990 END
```

FOR (See NEXT also)

Format

For scalar variable = expression TO expression [STEP exnression] ; i.e.,

```
FOR V = X TO Y STEP Z
```

Purpose

This statement – and the NEXT statement – are used to specify a loop. The FOR statement is used at the beginning of the loop; the NEXT statement at the end. The program lines in the range of the FOR statement are executed repeatedly, beginning with V = X; thereafter, V is incremented by Z until the value of V passes the limit specified by Y. The STEP portion of the statement may be positive or negative or may be omitted. If omitted, a step size of +1 is assumed. Loops may be nested with no limit.

If illegal values are assigned to the variables in a loop or if the loop designated by STEP is in the wrong direction or 0, the loop is executed once only and the program is continued. Examples of invalid values may be:

```
FOR R = 1 TO 10 STEP -1
FOR R = -1 TO -10 STEP 1
FOR R = 1 TO 10 STEP 0
```

A loop is executed to completion only if the values assigned the variables are valid. The following restrictions apply to the use of FOR loops

1. Branching into the range of a FOR loop from outside is not permissible (GO TO, GOSUB, IF-THEN).
2. Branching out of the range of a FOR loop is permissible, however, to conserve table storage, it should not be done repeatedly unless a subsequent normal terminating of an outer loop occurs or unless the loop is completely contained in a GOSUB routine. (GO TO, IF-THEN only, returning GOSUB branches are always legal).

Example

```
20 FOR X = 1 TO 50
30 PRINT X, SQR(X)
40 NEXT X
```

OR

```
20 FOR Z3=A(K) TO -COS(J) STEP -8 + INT(P(2))
30 R(Z3) = A(K) + A(Z3)
40 FOR Z4=R(Z3) TO A(K) : Q(Z4) =2*Z4*R(Z3)
50 PRINT Q(Z4),"VALUE",FN6(Q(Z4))
60 NEXT Z4: NEXT Z3
```

FOR (continued)

```
 50 GO TO 70
 60 FOR I = 1 TO 10 STEP 2               Illegal
 70 LET Z(I) = FNA(I)-LOG(I)             Branch
         .....                           FOR Loop
 90 NEXT I

100 FOR J = 1 TO 4
110 FOR K = 1 TO 6
         .....
120 IF Z(K)>10 THEN 160
         .....
150 NEXT K
160 NEXT J
                                         Proper branches
200 GOSUB 300                            out of a
         .....                           FOR Loop
300 FOR X = .1 TO Z STEP .05
         .....
340 IF A(I)<3.25 THEN 400
         .....
390 NEXT X
400 RETURN
```

FOR Loop within a GOSUB routine

GOSUB (See RETURN also)

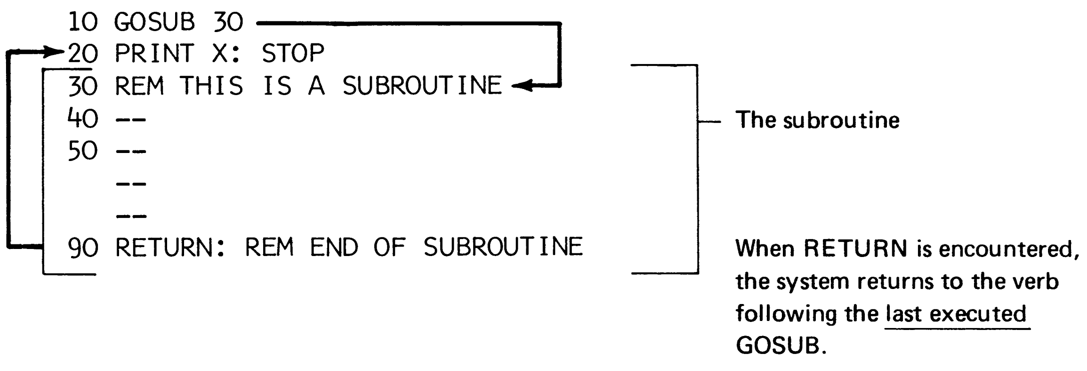
Format GOSUB line number

Purpose The GOSUB statement is used to specify a transfer to the first program line of a subroutine. This first program line may be any BASIC statement, including a REM statement to explain that this is the beginning of a subroutine. The logical end of the subroutine is a RETURN statement which directs the system to the verb following the last executed GOSUB; the RETURN statement must be the last executable statement on a line, but may be followed by non-executable statements as shown below:

```
120 X = 20: GOSUB 200: PRINT X
125 --
    --
    --
200 REM SUBROUTINE BEGINS
    --
    --
    --
210 RETURN: REM SUBROUTINE ENDS
```

The GOSUB statement may be used to perform a subroutine within a subroutine — a 'nested' GOSUB. This statement may not, however, be used to transfer within a FOR loop where a NEXT statement will be encountered before RETURN is encountered. Use of a GOSUB is not permitted in the immediate execution mode.

Example — Single GOSUB:



```
10 GOSUB 30
20 PRINT X: STOP
30 REM THIS IS A SUBROUTINE
40 --
50 --
   --
   --
90 RETURN: REM END OF SUBROUTINE
```

When RETURN is encountered, the system returns to the verb following the last executed GOSUB.

GOSUB (continued)

Example – Nested GOSUBS:

```
 10 GOSUB 30
 20 READ Q: STOP
 30 REM THIS IS A SUBROUTINE
 40 --
 50 --
    --
 70 GOSUB 150
 80 PRINT Q
 90 --
100 RETURN: REM END OF SUBROUTINE 30
110 --
    --
    --
150 REM THIS IS A NESTED SUBROUTINE
    --
    --      A 'nested' subroutine
    --
200 RETURN: REM END OF NESTED SUBROUTINE
```

The subroutine

A RETURN within the 'nested' subroutine directs the system back to the verb following the GOSUB for the 'nested' subroutine.

Example – Illegal GOSUB transfer into FOR loop

```
     500 GOSUB 750
         ....
         ....
FOR  700 FOR I = 20 TO 50
Loop     .....
     750 LET A(I) = LOG(12*A) - Z(I)
     760 NEXT I
     770 RETURN
```

NEXT statement occurs before RETURN

GO TO

Format	GO TO line number
Purpose	This statement is used to transfer to another area of the program. The GO TO directs the system to the line number where processing is to continue Use of this statement is not allowed in the immediate execution mode.
Example	`50 GO TO 10`

IF

Format

IF expression relational operator expression THEN line number.

where the relational operator is < , < = , = , > = , > , < > (3315 Teletype) or < , ⩽ , = , ≠ , > , ⩾ (3310 I/O Writer)

Purpose

This statement causes the system to skip the normal sequence of program lines and go to the line number following THEN, provided certain conditions are met. Quite simply, this may be described as a conditional GO TO statement, which compares the value of two expressions.

If the value of the first expression in the IF statement is in the specified relationship to the second expression, the system goes to the line number designated by THEN. If the specified relationship is not met, the program will continue in sequential order.

IF cannot be used in an immediate execution mode line.

Example

```
40 IF A < B THEN 35
50 IF A ≠ B THEN 70
```

INPUT

Format

INPUT variable $\left[\left\{ \text{, variable} \right\} \ldots \right]$

Purpose

This statement allows the current user to supply numeric data during the running of a program already stored in memory. For example, if the original programmer wants the user to supply the values for A and B while running the program, he enters, for example,

```
40 INPUT A, B
```

before the first program line which requires either of these values. When the system encounters this INPUT statement, it types the word INPUT and a colon on the following line and waits for the user to supply the two numbers. The program then continues. (It is common practice to identify requested variables by a preceding PRINT statement.)

Each number must be entered in the order in which it is to be used. If more than one number is entered on a line, each must be separated by a comma. Several lines may be used to enter the required INPUT data.

If there is a system-detected error in the entered data, the numbers must be re-entered beginning with the erroneous number. The numbers which precede the error are accepted.

A user may terminate an input sequence without supplying all the required input values by simply typing in a carriage return with no other information preceding it on the line. This will cause the system to immediately proceed to the next program statement. The INPUT list variables which have not received values will remain unchanged.

Example

```
10  INPUT A,B  ————→ Program Statement Executed
    INPUT      ————→ System Typeout
:  -3, 15      ————→ User response
```

LET

Format

$\left[\text{LET}\right]$ variable $\left[\left\{, \text{variable}\right\} \ldots\right]$ = expression

Purpose

The LET statement directs the system to perform certain computations and to assign the results to the variable or variables specified. The expression to the right of the equal sign is evaluated first.

The word LET is, however, optional. If it is omitted, its purpose is assumed.

Example

```
40 LET X(3), Z,Y=P+15/2+SIN(P-2.0)
```

OR

```
50 LET J=3
```

OR

```
10 X=A*E-Z*Y
```
→ Here LET is assumed

NEXT (See FOR also)

Format

NEXT scalar variable

Purpose

The NEXT statement signals the end of a loop begun with a FOR statement. The variable in the FOR statement and in its related NEXT statement must be the same.

During execution NEXT causes the referenced FOR statement index variable to be stepped. If the limit is not exceeded, transfer is made to the statement following the referenced FOR statement. If the limit is exceeded, the next sequential statement is executed.

In immediate execution mode, the NEXT statement and its corresponding FOR statement must both be in the immediate execution statement line.

Example

```
30 FOR M=2 TO N-1 STEP 30: J(M)=I(M)↑2
40 NEXT M
50 FOR X=8 TO 16 STEP 4
60 FOR A = 2 TO 6 STEP 2          ─┐
65 LET B(A,X) = B(X,A)              ├──►Nested Loop
70 NEXT A                          ─┘
80 NEXT X
```

PRINT

Format

PRINT print element [{ t print element } ....]

where a print element is an expression, TAB (expression), a text string in quotations or blank. t is a comma or semicolon.

Purpose

Printing may be done in zone-form which is signaled by a comma or packed form which is signaled by a semicolon.

Zone-form: PRINT print element [{, print element } ....]

The teletypewriter line is divided into 4 zones of 18 characters each, columns 0-17, 18-35, 36-53, 54-69. Each zone is 18 characters in length except for the last zone which is 16 characters in length. Selectrics will have two additional zones, columns 72-89, 90-107.

A comma signals that the next print element is to be printed starting in the next print zone, or if the final print zone is filled, then the first print zone of the next line.

For example:

```
10 X=214.230 :Y=3564 :Z=-.2379
20 PRINT X,Y,Z
```

will result in the print out of the following:

```
214.23            3564              -.2379
```

Packed-form: PRINT print element [{ ; print element } ....]

A semicolon signals that the next print element is to be printed immediately following the last print element, unless the last print element is an expression in which case a space is inserted between the value of the expression and the next print element.

For example:

```
10 X=2 :Y=3.4
20 PRINT "X=";X;"Y=";Y
```

will result in the print out of the following:

```
X= 2 Y=-3.4
```

A print statement can contain both comma and semicolon element separators. Each separator explicitly determines the amount of space skipped before the subsequent element is printed. A comma will cause advancement to the beginning of the next zone.

PRINT (continued)

A semicolon will cause 1 or no spaces to be skipped (depending upon whether the previous element was a variable or text string).

For example:

```
10 X=2 :Y=3 :Z=-4.2
20 PRINT "X=";X,"Y=";Y;"Z=";Z
```

will result in the following print out:

```
X= 2          Y= 3 Z=-4.2
```

The end of a print line signals a new line for output, unless the last symbol is a comma or semicolon. A comma signals that the next print element encountered in the program is to be printed in the next zone of the current line. A semicolon signals that the next print element is to be printed in the next available space, skipping 1 space if the last print element was an expression.

For example:

```
10 PRINT "X=";
20 PRINT 3.2970,
30 PRINT "Y=";64
```

will cause the following print out:

```
X= 3.297       Y= 64
```

PRINT with no print element advances the paper one line, or it causes the completion of a partially filled line.

Values of expressions are printed in one of two formats:

Format 1:	SM.MMMMMMME$\pm$XX	$10^{-1} >$ VALUE $\geq 10^{+8}$
Format 2:	SZZZZ.FFFF	$10^{-1} \leq$ VALUE $< 10^{+8}$

where	M	=	mantissa digits
	X	=	exponent digits
	F	=	fractional digits
	Z	=	integer digits
	S	=	minus sign of value < 0, or blank if value ≥ 0.

In format 2, the decimal point is inserted at the proper position or omitted if the value is an integer. Leading integer digit zeroes and trailing fractional digit zeroes are omitted.

PRINT (continued)

The following are examples of the printing of variables in the two formats:

```
Format 1:   2.3476214E-09
           -1.6472000E+22

Format 2:   23.479
             -.6374
            0
           -421
```

TAB (expression): This function permits you to specify tabulated formatting. For example, TAB (17) would cause the typewriter to move to column 17.

Positions are numbered 0 to 69 (3315 Teletype) and 0 to 107 (3310 I/O Writer). The value of the expression in the TAB function is computed, and the integer part is taken. The typewriter is then moved to this position. If it has already passed this position, the TAB is ignored. If the value of the expression is greater than 69 for the 3315 Teletype or greater than 107 for the 3310 I/O Writer, the typewriter will move to the beginning of the next line. Values of TAB expressions greater than 255 are illegal.

For example:

```
10  FOR I=1 TO 5
20  PRINT TAB(I);I
30  NEXT I
```

will cause the following print out:

```
1
 2
  3
   4
    5
```

RANDOM

Format

The RANDOM statement can be used in conjunction with the RND function to produce different sets of random numbers. For example, if RANDOM is the first instruction in a program using random numbers, then each time the program is executed a different set of random numbers will be used. Omitting the RANDOM statement will cause the "standard" set of random numbers to be used. Generally, when debugging a program which uses random numbers, the RANDOM statement is omitted so that the same results will be produced each time the program is executed.

Example

```
10 RANDOM
20 FOR I=1 TO 100
30 PRINT RND(X),
40 NEXT I
```

READ (See DATA and RESTORE also)

Format

READ variable $\left[\left\{, \text{variable}\right\} \ldots\right]$

Purpose

The READ statement is used to assign to variables the values contained in a DATA statement. Neither statement can be used without the other. The READ statement causes the variable(s) listed in it to be given, in order, the sequentially available numbers in the DATA statement(s). Numbers are retrieved from a DATA statement as they occur on that program line. If a READ statement references beyond the limit of existing numbers in a DATA statement, the system looks for another DATA statement in statement number sequence. If there are no more DATA statements in the program, an error message is written and the program is terminated.

READ statements customarily occur near the beginning or end of a program; DATA statements may be entered anyplace as long as they provide values in the correct order for the READ statements.

Example

```
100 READ A, B, C, A1, B1, C1
220 DATA 4, 315, -3.98, -174
230 DATA 5.62E - 3, -3.14158
```

REM

Format

REM text string

where 'text string' is any group of typewriter characters (excluding carriage return, colon, backspace, etc.)

Purpose

This statement provides a way to insert comments or explanatory remarks in a program. When the computer encounters a REM statement it ignores the remainder of the line – thus permitting the programmer to use the statement for his own purposes.

Example

```
200 REM THIS IS A SUBROUTINE FOR ENTERING DATA
210 REM ENTER DATA IN PRESCRIBED ORDER
220 REM THE FIRST NUMBER MUST BE N
```

RESTORE (See READ and DATA also)

Format

RESTORE [integer]

Purpose

The RESTORE statement allows the repetitive use of DATA statement values by READ statements. When RESTORE is encountered, the system returns to the nth DATA value; this value is that of the integer if one is included in the RESTORE statement; otherwise, it is assumed to be the first DATA number. Then when a subsequent READ statement occurs, the data will be read and used – beginning with the nth specified number – all over again.

Example

```
100 RESTORE ———————→ Start with the first DATA number.
     OR
100 RESTORE 11 ————→ Start with the 11th DATA number.
```

RETURN (See GOSUB also)

Format	RETURN
Purpose	The RETURN statement is used in a subroutine to return processing to the statement following the last executed GOSUB.
Example	See example for GOSUB.

STOP

Format

STOP

Purpose

The STOP statement causes the program execution to terminate. There can be several STOP statements within a program.

When STOP is encountered, the system types

```
3300 BASIC READY
:
```

to return the system to the user.

Example

```
100 STOP
```

TRACE

Format

$$\text{TRACE}\begin{Bmatrix}\text{ON}\\ \text{OFF}\end{Bmatrix}$$

Purpose

The TRACE statement provides for the tracing of the execution of a BASIC program. TRACE mode is turned on in a program when TRACE or TRACE ON is executed and turned off when TRACE OFF is executed. When the TRACE mode is on, printouts will be produced when:

Any program variable receives a new value during execution (LET, READ, FOR statements).

A program transfer is made to another sequence of statements (GO TO, GOSUB, IF)

For example:

```
30 LET X, Y, Z(5)=A+SIN(B)/C
```

will produce the TRACE printout:

```
X = Y = Z( ) = 29.631
```

For example:

```
40 READ A, B, C(22), D
```

will produce:

```
A = 9.4
B = 64.27
C = .37492000 E+11
D = 99.4
```

For example:

```
100 GO TO 200
```

will produce:

```
TRANSFER TO 0200
```

For example:

```
30 GOSUB 10
```

will produce:

```
TRANSFER TO 0010
```

TRACE (continued)

The following examples show the format printed for BASIC statements when the TRACE feature is being used:

LET: variable = variable = ... = received value
READ: variable = received value
FOR: variable = next index value
where 'received value' and 'next index value' are printed in the same format that is used in PRINT statements.

GO TO, GOSUB, IF: TRANSFER TO XXXX
where XXXX is a line number.

Example

```
100 TRACE
110 IF X = 15.1 THEN 150
120 LET X, Y  = 12.1 + 3
130 GO TO 110
140 TRACE OFF
150 STOP
RUN
```

Resulting printout:

```
X = Y = 15.1
TRANSFER TO 0110
TRANSFER TO 0150
3300 BASIC READY
:
```

EXTENDED BASIC STRUCTURE AND COMPONENTS

Wang 3300 EXTENDED BASIC is an extension of Standard BASIC and includes all the structure and components of Standard BASIC. It can be run on most 3300 systems but requires a slightly larger memory configuration than Standard BASIC. The language components unique to EXTENDED BASIC are described in this section. A summary of these features is presented below.

Summary of EXTENDED BASIC Features:

Alphanumeric String Variables and Literal Strings.

An additional function, STR, which permits substringing of string variables.

Fourteen Matrix Operation Statements including:

MAT Equality	MAT CON
MAT Addition	MAT ZER
MAT Subtraction	MAT IDN
MAT Multiplication	MAT READ
MAT Scalar Multiplication	MAT INPUT
MAT INV (Inverse, Determinant)	MAT PRINT
MAT TRN (Transpose)	MAT PRINT USING

Automatic Dimensioning and Redimensioning of Matrices. Program specifiable matrix redimensioning.

Nine data file operation statements applicable to Teletypes, cassettes, or disk:

FILES	IF END
FILEREAD	FILEEND
FILEWRITE	FILEMOD
MAT FILEREAD	FILESAVE
MAT FILEWRITE	

PRINT USING statement — to flexibly format printed variables and lines.

Conditional GO TO statement.

CHAIN statement — to automatically link and run program steps.

Alphanumeric String Variables

Wang 3300 Extended BASIC provides for an additional form of variable, the alphanumeric string variable. It is distinguished from numeric variables by the manner in which it is named, a letter followed by a $. String variables permit the user to input, process,

Alphanumeric String Variables (continued)

output and print alphanumeric strings of characters, (such as names, addresses and report titles). The following general forms are used to represent alphanumeric string variables:

Scalar string variable	
'letter' $	(i.e., A$, B$)
One-dimensional string array	
'letter' $ (d_1)	(i.e., A$(3),B$(N))
Two-dimensional string arrays	
'letter' $ (d_1,d_2)	(i.e., A$(2,3),B$(N,M))

Each string variable or string array element is initially assigned a value of 1 blank character. Thereafter, it can take the value and length of any alphanumeric character string up to 18 characters in length, which is received in a program statement. If a string variable receives a string value of less than 18 characters, it will reflect that shorter length in all subsequent operations until it receives another value.

For example,

```
10  LET A$="ABCD"
20  PRINT A$
```

will cause a print out of 4 characters: ABCD.

The dimensions of string arrays must be specified in a DIM or COM statement prior to their use in the program. They may, however, be redimensioned when used with the following matrix operation statements:

```
MAT INPUT, MAT READ, MAT FILEREAD.
```

(Refer to the section describing matrix operations.)

Alphanumeric Literal String

An alphanumeric literal string is a character string enclosed by double quotation marks. It is used in conjunction with string variables to provide a string value within a BASIC statement.

For example,

```
10  LET A$="ABCD"
20  IF B$ < "#XYZ" THEN 100
30  PRINT "NAME=";A$
```

Alphanumeric Literal String (continued)

Literal strings may be any length that can be expressed on one program line. However, when they are used in conjunction with string variables, they will be truncated to 18 characters, the maximum length permissible for string variables.

For example:

```
LET A$="ABCDEFGHIJKLMNOPQRST"
```

In this statement A$ will only receive the first 18 characters of the literal string, i.e., ABCDEFGHIJKLMNOPQR.

Statements in which String Variables are Permissible

Alphanumeric string variables can be used in the BASIC statements listed below. Literal strings can generally be used in place of string variables, except where a value is received.

```
LET                LET A$=B$(2)
                   A$="ABCD"
IF                 IF A$=B$  THEN 100
                   IF A$ <"DR"  THEN 200
                   IF "ABCD">B$  THEN 300
INPUT              INPUT A$,B$(4)
READ               READ C$,D$,E$(1,2)
FILEREAD           FILEREAD #0,A$,B$
MAT INPUT          MAT INPUT A$,B$                 (A$, B$ are arrays)
MAT READ           MAT READ A$,B$                  (A$, B$ are arrays)
MAT FILEREAD       MAT FILEREAD #0,A$,B$           (A$, B$ are arrays)
PRINT              PRINT A$,B$,"ABCD"
PRINT USING        PRINT USING 50,A$,B$,"LAST"
MAT PRINT USING    MAT PRINT USING 60,A$,B$        (A$, B$ are arrays)
FILEWRITE          FILEWRITE #2,A$,"GROUP1"
MAT FILEWRITE      MAT FILEWRITE #3,A$,B$          (A$, B$ are arrays)
DATA               DATA "ABCD", "EFGH"
```

STR (String Function)

Wang 3300 Extended BASIC provides a function which permits the user to extract, examine, compare or replace a specified portion of an alphanumeric string. The STR function operates on alphanumeric string variables, and can be used in any BASIC statement where alphanumeric variables are permissible. It has the following format:

STR (string variable, X_1 $\left[,X_2\right]$)

where X_1 = Starting character in string (an integer)

X_2 = Number of consecutive characters (an integer) (The specification of X_2 is optional)

STR (String Function) (continued)

For example,

```
STR(A$,3,4)
```

Means take the 3rd, 4th, 5th and 6th characters of A$.

```
STR(A$,3)
```

Means, starting with the 3rd character, take the remainder of the string A$.

When STR functions are used, they replace alphanumeric string variables and specify the portion of the string variable to be used. They may be used on either side of an equal sign or relation.

The following examples illustrate use of the string function:

Assuming B$="ABCDEFGH"

10 A$=STR(B$,2,4)	--- A$ is set to "BCDE".
20 STR(A$,4)=B$	--- Characters 4 through 18 of A$ are set to "ABCDEFGH".
30 STR(A$,3,3)=STR(B$,5,3)	--- The 3rd, 4th and 5th characters of A$ are set to "EFG".
40 IF STR(B$,3,2)="AB" THEN 100	--- Characters "CD" of B$ are compared to the literal string "AB".
50 READ STR(A$,9,9)	--- Characters 9 through 18 of A$ receive the next data statement value.

Matrix Operations

Wang 3300 Extended BASIC contains sixteen matrix operation statements. They allow the user to perform matrix algebra, input/output, and print arrays in a single program statement.

In general, the MAT (matrix) statements follow the conventions of matrix algebra. Single dimension arrays are treated as column vectors.

Matrix Operations

The MAT statements are generally similar to other BASIC statements but have the following unique syntax rules:

1. Each matrix statement begins with the word "MAT".
2. The variables used in matrix statements must always be array variables. They are always expressed without subscript notation.
3. A number of the MAT statements can result in a redimensioning of the receiving array variable. In some statements, it is done by supplying the new dimensions in the statement; in others, it is implicit.

The following matrix operation statements are available in Wang 3300 Extended BASIC:

MAT Addition	(Set Matrix = Matrix + Matrix)	-- `MAT A=B+C`
MAT CON	(Set Elements of Matrix = 1)	-- `MAT A=CON`
MAT Equality	(Set Matrix = Matrix)	-- `MAT A=B`
MAT IDN	(Set Matrix = Identity Matrix)	-- `MAT A=IDN`
MAT INV	(Set Matrix = Inverse of Matrix)	-- `MAT A=INV(B)`
MAT Multiplication	(Set Matrix = Matrix * Matrix)	-- `MAT A=B*C`
MAT READ	(Set Matrix = DATA values)	-- `MAT READ A,B`
MAT Scalar Multiplication	(Set Matrix = Scalar * Matrix)	-- `MAT A=(X)*B`
MAT Subtraction	(Set Matrix = Matrix – Matrix)	-- `MAT A=B-C`
MAT TRN	(Set Matrix = Transpose of Matrix)	-- `MAT A=TRN(B)`
MAT ZER	(Set Elements of Matrix = 0)	-- `MAT A=ZER`
MAT INPUT	(Receive Matrix Element from Terminal)	-- `MAT INPUT A,B`
MAT PRINT	(Print elements of a Matrix)	-- `MAT PRINT A,B`
MAT PRINT USING	(Print Matrix Using format statement)	-- `MAT PRINT USING 10,A,B`
MAT FILEREAD	(Input Matrix elements from a File)	-- `MAT FILEREAD #0,A,B`
MAT FILEWRITE	(Write Matrix elements onto a File)	-- `MAT FILEWRITE #0,A,B`

Array Dimensions

In both Standard and Extended BASIC, the dimensions of all arrays must be defined in a DIM or COM statement before they are used in a subscripted fashion in the program.

In Extended BASIC, however, if an array variable which has not been given any original dimensions is encountered in a MAT statement, it will automatically be defined as a two-dimensional array with dimensions of 10 by 10.

Array Redimensioning

There are several ways in which arrays can be redimensioned during program execution in Extended BASIC. A matrix may be redimensioned by appending new dimension(s), enclosed in parentheses, to array name(s) in any of the following MAT statements:

Matrix CON	example, `MAT A=CON(4,2)`
Matrix IDN	example, `MAT B=IDN(4,4)`
Matrix ZER	example, `MAT B=ZER(5)`
MAT INPUT	example, `MAT INPUT A(3,3)`
MAT READ	example, `MAT READ A(2,2),B(4,6)`
MAT FILEREAD	example, `MAT FILEREAD #0,A(10)`

In arithmetic matrix operations, the matrix on the left-hand side of the equal sign is automatically redimensioned, receiving the dimensions of the resulting matrix.

For example:

```
10  DIM A(20),B(2,2),C(2,2)
20  ...
30  ...
40  MAT A=B+C
```

The array A will be redimensioned from a vector of dimension 20 to a matrix of dimensions 2 by 2 by statement number 40.

It should be noted, however, that an error will result if a matrix is redimensioned to a size in which the resulting total number of elements is greater than the total number of elements specified by its original dimensions in the program. (Defined in a DIM or COM statement, or 10 by 10 if not specified.)

For example, if array A is originally given the dimensions of 10 by 10 by the following statement:

```
20 DIM A(10,10)
```

Then it can be redimensioned to:

(9,11),(8,12), . . . etc., or to a vector, (100).

However, redimensioning to the following dimensions is illegal:

(10,11),(9,12),(8,13),. . .   (101),(102),. . .

Matrix Arithmetic Restrictions

Multiple matrix operations are not permitted in a single statement. For example,

```
10  MAT C=A+B-D
```

$\longleftarrow$ illegal statement

is illegal. However, the following two statements can be used to achieve the desired results:

```
10  MAT C=A+B
20  MAT C=C-D
```

The same array variable name cannot appear on both sides of the equal sign in the following two matrix statements:

MAT Multiply
MAT TRN (Transposition)

Hence the following statements are illegal:

```
10  MAT C=C*A
20  MAT C=A*C
30  MAT C=C*C
40  MAT C=TRN(C)
```

} illegal statements

Data File Operations

Wang 3300 Extended BASIC has nine Data File operation statements:

FILES
FILEREAD
MAT FILEREAD
FILEWRITE
MAT FILEWRITE
IF END
FILEEND
FILEMOD
FILESAVE

FILEREAD statements allow a user to read data values from files on teletype paper tape, cassettes, or disk and directly assign them to BASIC program variables. Likewise, FILEWRITE statements allow the writing of variable values onto these devices, thus creating data files. The other file statements facilitate setting up, assignment, and termination of file operations in the program.

Data Files

Data files are sets of numeric and/or alphanumeric data which are written and stored on paper tape, cassettes or disk. They are created by the execution of one or more FILEWRITE statements in a program. A data file contains a continuous set of variable values resulting from the variables specified in the FILEWRITE statements. For example:

```
10  FILEWRITE #0, A,B1,C,D$
20  FILEWRITE #0, E,F$,G3,H,"FILE1"
30  FILESAVE #0
```

The above statement will create a paper tape file containing the current values of the variables A, B1, C, D$, E, F$, G3, H and the literal string "FILE1".

Data File Format

All data files have an identical format. Numeric variables will be stored in the Basic language format in which they appear in data statements, input statement entries, and PRINT statement outputs, (i.e., 23.421, 0, -10.2, 1.2345678E+27). Alphanumeric variables will be written exactly as they are stored, a string of alphanumeric characters, but they will be enclosed by quotation marks, (i.e., "ABCD", "JOHN C. JONES").

Cassette, disk, and paper tape files are generated by the system. Since files are in alphanumeric format, paper tape files can also be generated off-line on a teletype in the following manner:

Key in each variable followed by a CR/LF/RUBOUT/RUBOUT.
An end of file is designated by an X-OFF/CR/LF.
A data file will appear on tape in the following manner:

```
.LEADER CODE     (Blank Frames)
.VARIABLE #1     CR/LF/RUBOUT/RUBOUT
.VARIABLE #2     CR/LF/RUBOUT/RUBOUT
.
.
.VARIABLE #N     CR/LF/RUBOUT/RUBOUT
.X-OFF           CR/LF
.TRAILER CODE    (Blank Frames)
```

Logical File Designators and Logical Assignment

Files are always identified in a program by FILE designators. They consist of a # character followed by a number.

For example:	#1	(means file No. 1)
	#4	(means file No. 4)

Logical File Designators and Logical Assignment (continued)

In BASIC programs, #0 is always used to express the Teletype terminal paper tape reader and punch. The other logical files used in program statements (#1, #2, #3, etc.), are assigned either to a particular cassette tape or disk by a FILES statement. Assignment with a FILES statement is done in the program before any file operations are encountered.

For example:
```
10  FILES 3,4/2,"COSTFILE"
```

This statement would produce the following file assignments:

#1 =	Cassette No. 3
#2 =	Cassette No. 4, second file on the tape
#3 =	A disk file named "COSTFILE"

File Operation Restrictions

1. Up to 8 Disk or Cassette files can be assigned in a program, (#1 through #8).
2. An output file can be written and reread in the same program, but it must be first terminated by a FILEEND statement after it is written and before it is read.
3. Both numeric and alphanumeric data can be contained in the same file. However, when the data is read from the file, numeric data must be assigned to numeric variables, and alphanumeric data must be assigned to alphanumeric variables. If data is read in mixed mode, an error printout will result and the program will be terminated.
4. After all the data on a file is read, subsequent FILEREAD operations for that file will be ignored. The file will have an end of file status, which can be tested by an IF END statement.

For example, assume a paper tape file has the values 1, 2, 3, 4 on it:

```
10  FILEREAD #0,A,B,C,D  (Values read and assigned)
20  FILEREAD #0,E,F      (Statement ignored, no more data)
30  IF END #0 THEN 100   (End of File branch will occur)
```

EXTENDED BASIC STATEMENTS

This section contains a complete description of the additional statements available with EXTENDED BASIC. All STANDARD BASIC statements described in previous sections are also provided within EXTENDED BASIC and are functionally identical.

As in STANDARD BASIC; EXTENDED BASIC allows:

Multi-statements on a single line separated by a colon. For example,

```
100  MAT A=B+C :MAT PRINT A
```

The use of certain statements in immediate execution mode. The following EXTENDED BASIC statements can legally be used in immediate execution mode:

a) All matrix statements except:

```
MAT FILEREAD
MAT FILEWRITE
MAT INPUT
MAT PRINT USING
MAT READ
```

b) Chain.

Note – File operations are not legal in immediate execution mode.

CHAIN, CHAINR

Format

CHAIN[R][unit]

where unit = blank (Teletype paper tape reader)
1-16 (Cassette number)
"NAME" (Disk file)

Purpose

This is a BASIC program statement which in effect produces an automatic combination of the following:

STOP	(stop current program)
START or	(clear user area)
RESTART	(clear user area saving Common Data)
LOAD	(load new program)
RUN	(run it)

This permits segmented jobs to be run automatically without normal intervention. CHAINR is identical to CHAIN except the RESTART function is selected instead of START (i.e., common data is passed between programs).

CHAIN may be used in immediate execution mode; however, any statements following CHAIN in the immediate execution line will be ignored.

Example

```
500  CHAIN
500  CHAINR
500  CHAIN 3
500  CHAINR "PROGRAM2"
```

GOTO ON

Format

GOTO statement no. [{, statement no.} ...] ON (expression)

Purpose

Transfer is made to the ith statement specified in the list of statement numbers if the truncated integer value of the expression is i. If the value of the expression is less than 1, an error message is printed out and program execution is terminated. If the value of the expression is greater than the number of statements specified, transfer is made to the next statement.

Example

```
55  GOTO 100,120,130,140 ON (X*Y+2)
```

PRINT USING

Format

PRINT USING line number, print element [{t print element}...]

where a print element is an expression, string variable, or text literal string in quotation marks; line number is the line number of an image statement, and t is a comma or semicolon.

Purpose

PRINT USING operates in conjunction with a referenced image statement. It causes print elements in the PRINT USING statement to be formatted, inserted into the line image and printed. The image statement provides both alphanumeric text to be printed between the inserted print elements, and the format specifications for the inserted print elements. The format for each print element is made up of # characters to signify digits and optionally +, –, ., and ! characters to specify sign, decimal point position and exponent. For example:

```
10  X=2.3 :Y=-27.123
20  PRINT USING 30,X,Y
30  % ANGLE=##.##  LENGTH=+##.#
```

would result in the following printout:

```
ANGLE= 2.30   LENGTH=-27.1
```

If the number of print elements in the PRINT USING statement exceeds the number of formats specified in the image statement, the image statement is reused from the beginning for the remaining elements. If a comma follows the first remaining print element, a carriage return is issued before beginning the image statement again; if a semicolon follows the first remaining element, the carriage return is suppressed. The following examples illustrate comma and semicolon separators:

```
10  X=1 :Y=2 :Z=3
20  PRINT USING 30,X,Y,Z,      (comma separators)
30  % #.#

1.0                            (printout with
2.0                             comma separators)
3.0

10  X=1 :Y=2 :Z=3
20  PRINT USING 30,X;Y;Z;      (semicolon separators)
30  % #.#

1.0    2.0    3.0              (printout with
                                semicolon separators)
```

PRINT USING (continued)

Print Element Format Rules

An image statement variable format has the following general format:

$$\left\{\begin{matrix}\text{(no sign)}\\ +\\ -\end{matrix}\right\}\left[\#\#\#\#\ldots\right]\left[.\#\#\#\ldots\right]\left[!\,!\,!\,!\right]$$

It can be classified into three general formats:

Format 1	– Integer	i.e., ###, +####, -##
Format 2	– Fixed Point Number	i.e., ##.##, -#.###
Format 3	– Exponential	i.e., #.##!!!!, +.###!!!!

Print elements are formatted according to the following rules:

1. Numeric expression print elements:
   a) If the format specification contains a plus sign, the true sign of the expression is edited into the print line.
   b) If the format specification contains a minus sign, a blank for positive expressions or a minus sign for negative expressions is edited into the print line.
   c) If the format specification has no sign and the expression is negative, a minus sign is edited into the print line and the length of the format specification is increased by one.
   d) The expression value is printed according to the format specified in the image statement.

      Format 1 -- the integer part of the value is printed, truncating any fractions. Leading blanks are inserted.

      Format 2 -- the value is printed as a fixed point number, truncating or extending any fraction with zeros and inserting leading blanks according to the format specification.

      Format 3 -- the value of the expression is printed as a floating point number. The mantissa is formatted similar to a fixed point number in Format 2 (with leading blanks and trailing faction zeros). The exponent is always printed in the 4 character form: E±XX.

PRINT USING (continued)

e) If the length of the value to be printed is less than or equal to the length of the format specifications, the value is right-justified. If the length of the value is greater than the format specification, # characters are edited into the print line rather than the value to indicate the overflow.

2. Alphanumeric string variable or literal string print elements.

   The value of a string variable or a literal string in quotation marks is edited into the print line by replacing each character in the format specification (including sign, #, decimal point, and !) with the characters in the text string. The text string is left-justified. If the text string is shorter than the format specification, blanks are inserted on the right. The text string is truncated on the right if it is longer than the format specification.

Examples

For example:

```
100  PRINT USING 200,X,Y
200  % TOTAL SALES- ####    VALUE- $#####.##
```

(Printout)

```
TOTAL SALES- 1242       VALUE- $73694.23
```

For example:

```
50  PRINT USING 100,C↑2.1,E
100  % COEFFICIENT= +.###!!!!   ERROR= -##!!!!
```

(Printout)

```
COEFFICIENT= +.213E-04     ERROR= 23E-10
```

For example:

```
10  LET A=317.23
20  PRINT USING 30,A
30  % +#.##
```

(Printout)

```
#####
```
(Value too large for format)

For example:

```
10 LET A$="JOHN DOE"
20 PRINT USING 30,A$
30% SALESMAN: #########
```
(Printout)
```
SALESMAN: JOHN DOE
```

% – IMAGE STATEMENT (For Print Using)

Format

$\% \ t \ f \left[\left\{ t, f \right\} \ldots \right]$

where t is a text string (not containing any # characters) or null, and f is a format specification.

Purpose

This statement is used in conjunction with a PRINT USING statement to provide an image line for formatted output. The image statement contains text to be printed and format specifications used to format and insert print elements contained in the PRINT USING statement.

The image statement may have any printable characters of text inserted before and after print element format specifications. All text characters preceding the last format specification used by a print element will be printed. The image statement represents a character by character image of what will be printed. Each format specification is identified in an image statement by containing at least one # character. # characters may be preceded by a sign (+ or –) or a decimal point and followed by a decimal point or four ! characters, all of which are interpreted as part of the format. Format specifications including the sign, decimal point and exclamations cannot exceed 24 characters. The format specification has the following general form:

$$\begin{bmatrix} + \\ - \end{bmatrix} \left\{ \ldots \#\#\#\#.\#\#\#\# \ldots \right\} \quad \begin{bmatrix} !!!! \end{bmatrix}$$

This can be classified into the following three formats:

Format 1 -- an optional sign followed by one or more # characters. For example, –# # #

Format 2 -- an optional sign followed by one or more # characters, a decimal point, and zero or one or more # characters. For example, +# #.# # #

Format 3 -- an optional sign followed by one or more # characters, a decimal point, zero or one or more # characters, and four exclamation marks. For example, –# #. # # !!!!

Example

```
100  % TOTAL SALES= $###.##
110  % ##.###!!!!
120  % +#######
```

MATRIX OPERATIONS

MAT Addition

Format

MAT c = a + b

where c, a, and b are numeric array names.

Purpose

Adds two matrices or vectors of the same dimension. The sum is stored in array c. Array c may appear on both sides of the equal sign.

An error message will be printed and execution will be terminated if the dimensions of a and b are not the same. The resulting dimension of c is determined by the dimensions of a and b.

Example

```
10  DIM A(5,5),D(5,5),E(7),F(5),G(5)
20  MAT A = A + D
30  MAT E = F + G
40  MAT A = A + A
```

MAT CON Function

Format

MAT c = CON $\left[(d_1, \left[d_2\right])\right]$

where c is a numeric array name and d_1, d_2 are expressions specifying new dimensions.

Purpose

This statement sets all elements of the specified matrix to one (1). Using (d_1, d_2) causes the matrix to be redimensioned to these dimensions. If (d_1, d_2) is not used, the matrix has the dimensions specified in a previous DIM statement or as redimensioned in a previous MAT instruction.

Example

```
10  MAT A=CON(10)
15  MAT C=CON(5,7)
20  MAT B=CON(5↑Q,S)
30  MAT A=CON
```

MAT Equality

Format

MAT a = b

where a and b are numeric array names.

Purpose

This statement replaces each element of array a with a corresponding element of array b. Array a is redimensioned to conform to the dimensions of array b. Multiple equal statements such as MAT A,B=C are not allowed.

Example

```
10  DIM A(3,5),B(3,5)
20  MAT A=B

30  DIM C(4,6),D(2,4)
40  MAT C=D

50  DIM E(6),F(7)
60  MAT F=E
```

MAT IDN

Format

MAT c = IDN [(d1 [,d2])]
where c is a numeric array name and d_1, d_2 are expressions specifying new dimensions.

Purpose

This statement causes the specified matrix to assume the form of the identity matrix. If the specified matrix is not a square matrix, an error message is printed and execution is terminated.

Using (d_1, d_2) causes the matrix to be redimensioned to these dimensions. If (d_1, d_2) is not used, the matrix has the dimensions specified in a previous DIM statement or as redimensioned in a previous MAT instruction.

Example

```
10  MAT A=IDN(4,4)
20  MAT B=IDN
30  MAT C=IDN(X,Y)
```

MAT INPUT

Format

MAT INPUT array name $\left[(d_1 \left[,d_2\right])\right] \left[,\{\text{array name} \left[(d_m \left[,d_n\right])\right]\}\ldots\right]$

where d is an expression specifying a new dimension. Array name is numeric or alphanumeric array.

Purpose

The MAT INPUT statement allows the user to supply values via the terminal for an array(s) during the running of a program. When the system encounters a MAT INPUT statement, it prints the word INPUT followed by a colon and waits for the user to supply values for the arrays specified in the MAT INPUT statement. The dimensions of the array(s) are assumed to be as last specified in the program (by a DIM statement or as redimensioned in another MAT statement) unless the user redimensions the array(s) by specifying the new dimension(s) after the array name(s).

The values input are assigned to an array row by row until the array is filled. If more than one value is entered on a line, the values must be separated by a comma. Several lines may be used to enter the required data. Excess data is ignored. If there is a system-detected error in the entered data, the data must be re-entered beginning with the erroneous number. The data which precedes the error is accepted.

Entering no data on an input line (only a carriage return), signals the system to ignore the remaining elements of the array currently being filled. When alphanumeric string variable arrays are used in the list, each element is reset to the length of the received string. The input data must be compatible with the array, (i.e., numeric input for numeric array, alphanumeric literal strings input for alphanumeric arrays).

Example

```
05  DIM A(2)
10  MAT INPUT A,B(2),C(2,4)

INPUT:   -3, -5, .612, .41       (Values for A(1),A(2),B(1),B(2))
:  -6.2, -5.6, 98.               (Values for C(1,1),C(1,2),C(1,3))
:  carriage return               (Ignore rest of input for C)
```

With String Variable Arrays:

```
10  DIM C$(2)
100  MAT INPUT A$(4),B(2),C$

INPUT: "RAD", "DEG", "MIN", "SEC", 2.5, 5.6
:  "LAST RESULT", "ROTATE"
```

MAT INVerse and the DET Function

Format

MAT c = INV(a)

where c and a are numeric array names.

Purpose

This statement causes matrix c to be replaced by the inverse of matrix a. Array c may appear on both sides of the equal sign. The dimensions of c are determined by the dimensions of a. Matrix a must be a square, non-singular matrix; otherwise, an error message is printed and the program execution is terminated.

After inversion, the DET function equals the determinant of matrix a.

Example

```
10  DIM E(5,5), F(6,6)
    ...
    ...
200 MAT F=INV(E)
300 PRINT "DET OF E =", DET
```

Error Condition: (Matrix singular)

```
10 MAT E=ZER(3,3)
20  MAT F=INV(E)
               ↑ERROR 42
```

MAT Multiplication

Format

MAT c = a * b

where c, a, and b are numeric array names.

Purpose

The product of arrays a and b is stored in array c. Array c may not appear on both sides of the equal sign. If the number of columns in matrix a does not equal the number of rows in matrix b, an error message will be printed and execution will be terminated. The resulting dimension of c is determined by the number of rows in a and number of columns in b.

Example

```
10  DIM A(5,2),B(2,3),C(4,7)
20  DIM E(3,4),F(4,7),G(3,7)
40  MAT G=E*F
50  MAT C=A*B
```

Error Condition: (Row and columns not compatible)

```
10  DIM A(2,2),B(4,4)
20  MAT C=A*B
             ↑ERROR 45
```

MAT PRINT

Format

MAT PRINT array name [t] [{array name [t]} . . .]
where t is a comma or semicolon.

Purpose

The MAT PRINT statement prints the arrays in the order listed in the statement. Each matrix is printed row by row. All the elements of a row are printed on many lines as required with single line spacing. A blank line is used to separate rows. The first element of a row always starts at the beginning of a new print line. A matrix is printed in zone-form (4 values per line) unless the array name is followed by a semicolon, in which case the array is printed in packed-form (see PRINT). A vector (1-dimensional array) is printed as a column vector (double spaced).

Example

```
10  DIM A(4),B(2,4),B$(10),C$(6)
100  MAT PRINT A; B, C$
200  MAT PRINT A, B$;
```

MAT PRINT USING

Format

MAT PRINT USING line number, array variable $\left[\left\{\text{t array var.}\right\}\ldots\right][\text{t}]$

where line number is the statement number of an image statement and t is a comma or semicolon.

Purpose

This statement allows the formatted printing of arrays. It incorporates the functions of the MAT PRINT and PRINT USING statements. The array elements are printed in the format(s) contained in the image statement referenced in the MAT PRINT USING statement. The image statement is reused in order to output all the elements of the matrix. A trailing comma after the array variable specifies that a carriage return will only be output when the image statement is used up and must be restarted. For most applications, a trailing semicolon will be used. With a trailing semicolon (or blank), each matrix will be printed out row by row with all elements of a row printed out on as many lines as required with single line spacing. An additional blank line is used to separate rows. At the beginning of each row, the image statement is restarted. When trailing semicolons are used, rows of matrices, with more than four elements, can be printed on a single line in any specified format.

Example

For example:

```
10  DIM A(2,5)
20  MAT FILEREAD #0,A
30  MAT PRINT USING 40,A;
40% +#.#!!!!
```

(Printout)

```
+6.2E-07  -1.3E-17  +7.3E-11 +1.2E-08 -3.9E-08

-4.9E-08  +6.9E-10  +8.4E-09 +5.2E-14 -5.4E-13
```

For example:

```
10  DIM B(3,2)
...
...
50  MAT PRINT USING 60,B
60% ANGLE= ###.## DEG    ERROR= ##.##
```

(Printout)

```
ANGLE=  36.49 DEG   ERROR=   5.21

ANGLE=  85.60 DEG   ERROR=  11.92

ANGLE= 135.51 DEG   ERROR=   8.61
```

MAT READ

Format

$$\text{MAT READ array name} \left[(d_1 \left[,d_2\right])\right] \left[\left\{, \text{array name} \left[(d_m \left[,d_n\right])\right]\right\} \ldots\right]$$

where d is an expression specifying a new dimension. Array name is a numeric or alphanumeric array.

Purpose

The MAT READ statement is used to assign values contained in DATA statements to array variables without referencing each member of the array individually. The MAT READ statement causes the arrays listed in it to be filled, in order, with the sequentially available values in the DATA statement(s). Each array is filled row by row. Values are retrieved from a DATA statement in the order they occur on that program line. If a MAT READ statement references beyond the limit of existing numbers in a DATA statement, the system searches for the next sequential DATA statement. If there are no more DATA statements in the program, an error message is printed out and execution is terminated.

Alphanumeric string variable arrays can also be used in the list. The information entered in the data statements must be compatible with the array (i.e., numeric values for numeric arrays, alphanumeric literal strings for alphanumeric arrays).

The dimensions of the array(s) are assumed to be as last specified in the program (by a DIM statement or as redimensioned in another MAT statement) unless the user redimensions the array(s) by specifying the new dimension(s) after the array name(s).

Example

```
10  MAT READ A,B(2,3),C(9)
100  DATA 1,-.2,315,-.398,6.21,0,0
110  DATA 4,6,-.1201,5.62E-3,8.1,-.101
120  DATA .12,-1.7,0,-1,1,9,.871,4,-1


10  DIM A(2,2),B$(3,2)
20  MAT READ A,B$,C(2),D$(4)
100 DATA 1,2.3,-3.4E12,5
110 DATA "ABC", "DEFG", "HI", "J", "KL", "MNO"
120 DATA .2345,1E-12,"AB","CD","EFGH","IJK"
```

MAT Scalar Multiplication

Format

MAT c = (k) * a

where c and a are numeric array names and k is an expression.

Purpose

Each element of matrix or vector a is multiplied by the value of expression k and the product is stored in array c. Array c may appear on both sides of the equal sign. The dimensions of c are determined by the dimensions of a.

Example

```
20  MAT  C=(SIN(X))*A
30  MAT  D=(X+Y↑2)*A
40  MAT  A=(5)*A
```

MAT Subtraction

Format

MAT c = a – b

where a, b, and c are numeric array names.

Purpose

Subtract matrices or vectors of the same dimension. The difference of each element is stored in the corresponding element of c. Array c may appear on both sides of the equal sign.

An error message will be printed and execution will be terminated if the dimensions of a and b are not the same. The resulting dimension of c is determined by the dimensions of a and b.

Example

```
10  DIM A(6,3),B(6,3),C(6,3),D(4),E(4)
20  MAT C = A - B
30  MAT C = A - C
40  MAT D = D - E
```

MAT TRN (Transpose)

Format

MAT c = TRN(a)

where a and c are numeric array names.

Purpose

This statement causes array c to be replaced by the transpose of array a. The resulting dimensions of c are determined by the transpose of a. Array c may not appear on both sides of the equal sign.

Example

```
10  MAT C=TRN(A)
```

MAT ZER Function

Format

MAT c = ZER [(d1 [, d2])]

where c is a numeric array name and d_1, d_2 are expressions specifying new dimensions.

Purpose

This statement sets all elements of the specified matrix to zero. Using (d_1, d_2) causes the matrix to be redimensioned. If (d_1, d_2) is not used, the matrix has the dimensions specified in a previous DIM statement or as redimensioned in a previous MAT instruction.

Example

```
10  MAT C=ZER(5,2)
20  MAT B=ZER
30  MAT A=ZER(F,T+2)
40  MAT D=ZER(20)
```

DATA FILE OPERATIONS

IF END

Format

IF END #N THEN statement number

where N is the file designator number.

Purpose

The IF END statement is used to test for end of file condition following FILEREAD and MAT FILEREAD statements. If an end of file was encountered on the last previous FILEREAD or MAT FILEREAD statement for the designated file, then the IF END statement will cause a transfer to be made to the specified statement number.

Example

```
100  FILEREAD #2, A, B, C
110  IF END #2 THEN 130
```

FILEEND

Format

FILEEND #N

where N is the file designator number.

Purpose

The FILEEND statement is used to close or terminate an output file that has just been written and must be reread in the same program. It is also used to bypass the remainder of an input file.

The FILEEND statement causes the following action to occur for each type of file and unit:

Output Paper Tape	An end of file is punched followed by 50 blank frames.
Output Cassette Tape	An end of file sequence is written and the tape is rewound and positioned at the start of the file.
Output Disk	An end of file is written and the file is redefined as an input file. The file pointer is reset to the 1st file element.
Input Paper Tape, Cassette, Disk	The remainder of the file is read and ignored until an end of file is encountered.

Example

```
300  FILEEND #0
350  FILEEND #2
```

FILEMOD

Format

FILEMOD #N, $\left\{ \begin{matrix} \text{SCRATCH} \\ \text{"NAME"} \end{matrix} \right\}$

where N is the file designator number

Purpose

FILEMOD is used to remove a file catalogued on disk or to rename it. If the word SCRATCH appears at the end of the statement, the file is permanently removed from the disk. It can be renamed by listing the new name enclosed by quotation marks.

Example

```
100  FILEMOD #2, SCRATCH
200  FILEMOD #4, "FILE-C"
```

FILEREAD

Format

FILEREAD #N, variable, variable, . . .

where N is the file designator, (#1, #2, etc.) and variable is any numeric or alphanumeric string variable or array variable element.

Purpose

The FILEREAD statement causes data to be read from the designated file and sequentially assigned to each variable in the list. Both numeric and alphanumeric data may be read in the same FILEREAD statement, but the data assigned to each variable must correspond to the variable type (i.e., numeric data for numeric variables and alphanumeric data for alphanumeric variables).

Values will be successively read from the file until all variables in the list are satisfied. If a file is exhausted before all variables in the list are satisfied, the remaining variables in the list are ignored and successive IF END statements for the file will cause branching.

Example

```
20  FILEREAD #0,A,B,C$,D(1,2),E$(4,1)
```

where the file might contain:

```
-3.249, 1.2E+20, "ABCD", 24.6, "EFGH"
```

FILES

Format

$$\text{FILES}\ \underbrace{\text{unit}\ \left[/\text{m}\right]}_{\text{FILE \#1}}\ \left[\underbrace{,\text{unit}\ \left[/\text{m}\right]}_{\text{FILE \#2}}\ \ldots\right].$$

For Disk Files:

UNIT is the name of a disk file (i.e., "MASTER","FILE34"); m is the number of disk storage units required for a new output file (i.e., 10 means 10 times 1024 characters).

For Cassette Files:

UNIT is the logical tape number (1-32) (i.e., 1 = Cassette No. 1, 2 = Cassette No. 2); m is the file location on the tape (i.e., 3/4 means Cassette No. 3 and the 4th file on tape).

For Paper Tape Files:

The file designator #0 when used in a program always implies the teletype terminal paper tape reader and punch. Therefore, a FILES statement is not necessary for paper tape files.

Purpose

The FILES statement is used to assign a physical unit to each file designator reference in the program. All file designators used in a program must contain a corresponding unit assignment in the FILES statement list. The files are assigned sequentially, the first item in the list assigns file #1, the second item assigns file #2, etc.

The file statement must appear in the program before any file operations. It may, however, be modified after a program is loaded and before it is run. This permits a program to be written with file operations; the actual units to be used for the files can be assigned at execution time.

Example

```
10 FILES 3,5/2,"SAM","JOE"/10
```

makes the following assignments:

FILE #1 is Cassette Tape 3 (1st file).
FILE #2 is Cassette Tape 5, 2nd file on tape.
FILE #3 is a Disk input file already catalogued under the name "SAM".
FILE #4 is a new Disk output file which will be catalogued under the name "JOE". It will be allocated 10 (1024) characters of storage.

FILESAVE

Format

FILESAVE #N

where N is the file designator number.

Purpose

FILESAVE is used when an output file just written is to be permanently saved.

FILESAVE causes the following action to be taken for each unit:

Paper Tape	An end of file and 50 blank frames are punched.
Cassette Tape	An end of file is written and the tape remains positioned where it is. (Another file can subsequently be written.)
Disk	The file is stored and catalogued on Disk.

Example

```
200  FILESAVE #2
```

FILEWRITE

Format

FILEWRITE #N, file element $\left[\left\{ \text{, file element} \right\} \ldots \right]$

where N is the file designator (i.e., #1, #2, etc.) and file element is any legal numeric or alphanumeric variable, numeric or alphanumeric array element, alphanumeric literal, or numeric expression.

Purpose

The FILEWRITE statement causes the numeric values or alphanumeric character strings resulting from each file element to be sequentially written onto the file designated. Numeric values will be written in a format identical to that resulting from the PRINT statement:

Format 1:	SM.MMMMMMME±XX	$10^{-1} >$ value $\geq 10^{+8}$
Format 2:	SZZZZ.FFFF	$10^{-1} \leq$ value $< 10^{+8}$

Alphanumeric variables will be written identically to the character string data they contain, except that quotation marks will be inserted immediately before and after the string.

Example

```
100  FILEWRITE #2,A,B(2),C$,D$(1,2),"ABCD",SIN(.25)
```

where the values written might be the following:

```
1.5, -2.1642E+24, "MNOP", "QRS", "ABCD", .23648784
```

MAT FILEREAD

Format MAT FILEREAD #N, array name $\left[(d_1 \left[, d_2\right]) \right] \left[\left\{ \text{array name} \left[(d_1 \left[, d_2\right]) \right] \right\} \ldots \right]$

where N is the file designator number of the file being referenced, and d_1 and d_2 are new dimensions for the array.

Purpose

This statement is logically equivalent to FILEREAD, except that each variable listed is an array variable. Each file element in each array will be assigned a value from the file (row by row). If optional dimensions appear in the list following the variable name, the array will be automatically redimensioned to the new dimensions. If the file is exhausted before all the arrays are filled, the remaining elements of the array currently being worked on and all the elements of the remaining arrays are left unchanged and successive IF END statements for the file will cause branching. Alpha/numeric arrays may be used in the list. The data received from the file must, however, be compatible (i.e., numeric data for numeric arrays, alpha/numeric data for alpha/numeric arrays).

Example

```
100  MAT FILEREAD #2, A, B, C

200  MAT FILEREAD #1, A$, B, C$(4,6)
```

MAT FILEWRITE

Format

MAT FILEWRITE #N, array name [{, array name} . . . .]

where N is the file designator number of the file being referenced, and array name is any numeric or alphanumeric array name.

Purpose

The statement MAT FILEWRITE causes the values of the specified arrays to be written on the referenced file without referencing each member individually. Each array is written row by row with values separated by commas. Numeric values are written in a format identical to that used for PRINT (i.e., fixed or exponential format), and alphanumeric values are written as character strings enclosed by quotation marks.

Example

```
100  MAT FILEWRITE #0, A, B, C$

200  MAT FILEWRITE #4, A, B$, C
```

BASIC TERMINAL OPERATING INSTRUCTIONS

The standard terminals for WANG BASIC are the 3310 I/O Writer and the 3315 Teletype.

Initial Start Procedures

The first steps in entering a program into WANG 3300 BASIC are the turning on of the terminal and signaling the system that the user is ready to communicate with BASIC. These steps are:

1. Set the ON/OFF switch to ON on the 3310 I/O Writer or set the LINE/OFF/LOCAL switch to LINE on the 3315 Teletype.
2. Press the ATTENTION key on the 3310 I/O Writer or the ESCAPE (ESC) key on the 3315 Teletype.

The system recognizes that a new user is on the air and replies with:

```
 3300 BASIC READY
:
```

'3300 BASIC READY' is the system reply; the colon on the following line indicates that the system belongs to the user and he may proceed with entering his program and commands.

The first command should be START to remove any programs which may have been entered at an earlier time.

Deleting Incorrect Characters

The user may correct single characters in a line by using the backspace key on the 3310 I/O Writer or the backarrow on the 3315 Teletype. This action 'crosses out' the most recently typed characters. Several characters may be so crossed out in succession, by sequential backspaces or backarrows.

On the 3310 I/O Writer, the backspace correction sequence is completed by manually turning the carriage to advance the paper to the next line; the corrected text may then be entered.

```
             ←←←  ———→ 3 backspaces
:340  LET Q=X+Y2
             -Y2  ———→ user correction
```

BASIC TERMINAL OPERATING INSTRUCTIONS (continued)

On the 3315 Teletype, the user enters the corrected text immediately following the backarrows as shown in the following example:

```
:310 LET Q = X + Y2 ←←←-Y2
                       └─►user correction
:310 LET Q = X - Y2 ──────►line as received by system
```

Deleting the Current Line

If the user detects an error in a program text line he has just entered, he may type a degree character, 'o', on the 3310 I/O Writer or a backslash character, '\', on the 3315 Teletype. This character will cause the line to be ignored. When the 'o', or '\' is encountered, the system will index to the next line and type a colon requesting subsequent input. For example,

```
:120 LET X = AB/C\          (3315 Teletype)
:
```

or

```
:120 LET X = AB/C °         (3310 I/O Writer)
:
```

The user may then re-enter the correct line.

Deleting a Previously Entered Line

The user may delete a program statement line he entered early in his program by typing the line number of the program statement line to be deleted immediately followed by a carriage return.

This correction procedure applies only to statement lines; system commands and immediate mode execution lines cause immediate action once they are entered and are automatically deleted.

Requests for Execution

After the user has entered his program, he requests its execution by entering a RUN or RERUN command. The RUN command causes the entire program to be run from its beginning and all variables are set to initial values of zero. RERUN, on the other hand, does not reinitiate values (it leaves them at the current setting). The RERUN command may be used to run a portion of a program; e.g., RERUN 60 means to rerun the program beginning

BASIC TERMINAL OPERATING INSTRUCTIONS (continued)

with program line 60. If the RERUN command is entered without a line number, the program is executed from the beginning.

Breaking Program Execution

The user may interrupt the BASIC system while his program is running or printing. He merely presses the ATTENTION key on the 3310 I/O Writer or the ESCAPE (ESC) key on the 3315 Teletype. The system then stops the program execution or listing and types out:

```
  3300 BASIC READY
:
```

and thus returns the system to the user.

Initiating for New Program Entry

A BASIC program is initiated by entry of a START or RESTART command. The START command initiates the entire user program area, including the removal of all previously stored program text. The RESTART command initiates the user program area but does not remove variable data that is designated to be used by successive programs, (common variables).

BASIC ERROR MESSAGES

The WANG BASIC system of error detection provides for the reporting of errors as they occur. The user may then correct the error before proceeding with his program.

When an error is detected, the text line being scanned by the system is typed (if it is not already on the typewriter sheet) and, on the next line, an up-arrow is placed at the point of the current scan, followed by the error message number.

The following line shows this format:

```
:10  DIM A(P)
           ↑ ERR 13
```

The user may then refer to the following table of error messages to identify the error by code number. The table contains a description of each error and a suggested correction procedure.

Code 01

Error:	TEXT OVERFLOW
Cause:	All available space for BASIC statements and system commands has been used.
Action:	Shorten and/or chain program by using COM statements, and continue. The compiler automatically removes the current and highest-numbered statement.

Example:

```
:10 FOR I=1 TO 10
:20 LET X=SIN(I)
:30 NEXT I
      ....
      ....
      ....
:820 IF Z=A-B THEN 900
↑ERR 01
```

(the number of characters in the program exceeded the text table limit when line 820 was entered)

User must shorten and segment program.

Code 02

Error:	TABLE OVERFLOW
Cause:	All available space for internal compiler tables has been used (storage of variables, values, etc.) or An endless program loop was encountered, or Illegal use of DEF FN statements (endless loop).
Action:	Shorten or correct and/or chain the program by using COM statements and continue.

Example:

```
:10 DIM A(19),B(10,10),C(10,10)
:RUN
↑ERR 02
```

(the table space required for variables exceeded the table limit for variable storage as line 10 was processed)

User must compress program and variable storage requirements.

Code 03

Error:	MISSING COMMA
Cause:	A comma (,) was expected.
Action:	Correct statement text.

Example:

```
:10 LET STR(A$;6,4) = "ABCD"
               ↑ERR 03
:10 LET STR(A$,6,4) = "ABCD"                (Possible Correction)
```

Code 04

Error:	MISSING LEFT PARENTHESIS
Cause:	A left parenthesis (() was expected.
Action:	Correct statement text.

Example:

```
:10 DEF FNA V)=SIN(3*V-1)
           ↑ERR 04
:10 DEF FNA(V)=SIN(3*V-1)                   (Possible Correction)
```

Code 05

Error: MISSING RIGHT PARENTHESIS

Cause: A right ()) parenthesis was expected.

Action: Correct statement text.

Example:

```
:10 Y=INT(1.2↑5
               ↑ERR 05
:10 Y=INT(1.2↑5)                    (Possible Correction)
```

Code 06

Error: MISSING EQUALS SIGN

Cause: An equals sign(=) was expected.

Action: Correct statement text.

Example:

```
:10 DEF FNC(V)-V+2
              ↑ERR 06
:10 DEF FNC(V)=V+2                  (Possible Correction)
```

Code 07

Error: MISSING QUOTATION MARKS

Cause: Quotation marks (") were expected.

Action: Correct statement text.

Example:

```
:10 PRINT "ERROR
                ↑ERR 07
:10 PRINT "ERROR"                   (Possible Correction)
```

Code 08

Error: MISSING NUMBER SIGN

Cause: A number sign (#) was expected.

Action: Correct statement text.

Example:

```
:10 FILEREAD 0,A,B
             ↑ ERR 08
:10 FILEREAD #0,A,B                 (Possible Correction)
```

Code 09

Error: MISSING ASTERISK

Cause: An asterisk (*) was expected.

Action: Correct statement text.

Example:

```
:10 MAT A=(3)B
             ↑ERR 09
:10 MAT A=(3)*B                     (Possible Correction)
```

Code 10

Error: INCOMPLETE STATEMENT

Cause: The end of the statement was expected.

Action: Complete the statement text.

Example:

```
:10 PRINT X"
          ↑ERR 10
:10 PRINT "X"                    (Possible Correction)
     OR
:10 PRINT X
```

Code 11

Error: MISSING LINE NUMBER

Cause: The line number is missing or a referenced line number is undefined.

Action: Correct statement text.

Example:

```
:10 GOSUB ZOO
         ↑ERR 11
:10 GOSUB 200                    (Possible Correction)
```

Code 12

Error: MISSING STATEMENT TEXT

Cause: The required statement text is missing (THEN, STEP, etc.)

Action: Correct statement text.

Example:

```
:10 IF I=12*X,45
            ↑ERR 12
:10 IF I=12*X THEN 45            (Possible Correction)
```

Code 13

Error: MISSING OR ILLEGAL INTEGER

Cause: A positive integer was expected or an integer was found which exceeded the allowed limit.

Action: Correct statement text.

Example:

```
:10 COM D(P)
         ↑ERR 13
:10 COM D(8)                     (Possible Correction)
```

Code 14

Error: MISSING OPERATOR

Cause: A relational operator ($<$, $\leq$, $=$, $\neq$, $\geq$, $>$) was expected.

Action: Correct statement text.

Example:

```
:10 IF A-B THEN 100
          ↑ERR 14
:10 IF A=B THEN 100              (Possible Correction)
```

Code 15

Error:	MISSING EXPRESSION
Cause:	A variable, or number, or a function was expected.
Action:	Correct statement text.
Example:	

```
:10 FOR I=, TO 2
          ↑ ERR 15
:10 FOR I=1 TO 2                        (Possible Correction)
```

Code 16

Error:	ILLEGAL USE OF A VARIABLE
Cause:	A scalar variable was expected.
Action:	Correct statement text.
Example:	

```
:10 FOR A(3)=1 TO 2
          ↑ ERR 16
:10 FOR B=1 TO 2                        (Possible Correction)
```

Code 17

Error:	ILLEGAL USE OF A VARIABLE
Cause:	An array variable was expected.
Action:	Correct statement text.
Example:	

```
:10 DIM A2
         ↑ERR 17
:10 DIM A(2)                            (Possible Correction)
```

Code 18

Error:	ILLEGAL USE OF A VARIABLE
Cause:	An alphanumeric variable was expected.
Action:	Correct statement text
Example:	

```
:10 LET A$=B
           ↑ERR 18
:10 LET A$=B$                           (Possible Correction)
```

Code 19

Error:	MISSING NUMBER
Cause:	A number was expected.
Action:	Correct statement text.
Example:	

```
:10 DATA L
         ↑ERR 19
:10 DATA 1                              (Possible Correction)
```

Code 20

Error: ILLEGAL NUMBER FORMAT

Cause: A number format is illegal.

Action: Correct statement text.

Example:

```
:10 A=12345678.23                    (More than 8 digits of mantissa)
                ↑ERR 20
:10 A=12345678                       (Possible Correction)
```

Code 21

Error: MISSING LETTER OR DIGIT

Cause: A letter or digit was expected.

Action: Correct statement text.

Example:

```
:10 DEF FN.(X)=X↑5-1
           ↑ERR 21
:10 DEF FN1(X)=X↑5-1                 (Possible Correction)
```

Code 22

Error: ILLEGAL VARIABLE ATTRIBUTE USAGE

Cause: A variable was referenced as both a scalar and an array variable; or an array variable was not referenced as previously defined in this program or as a common variable in another program (i.e., an array variable has been referenced both as a 1-dimensional and as a 2-dimensional array).

Action: Correct statement text.

Example:

```
:10 DIM A(5,5)
:20 A=3
    ....
    ....
    ....
:90 END
:RUN

 20 A=3
    ↑ERR 22
:20 B=3                              (Possible Correction)
```

Code 23

Error: NO PROGRAM STATEMENTS

Cause: A RUN or RERUN command was entered but there are no program statements.

Action: Enter program statements.

Example:

```
:START
:RUN
 ↑ERR 23
```

Code 24

Error: ILLEGAL IMMEDIATE MODE STATEMENT

Cause: An illegal verb or transfer in an immediate execution statement was encountered.

Action: Re-enter a corrected immediate execution statement.

Example:

```
:10 PRINT "A=";A
:A=1 :GO TO 10

 A=1 :GO TO 10
 ↑ERR 24
```

Code 25

Error: ILLEGAL GOSUB/RETURN USAGE

Cause: There is no companion GOSUB statement for a RETURN statement, or a RERUN starting location is illegal.

Action: Repeat execution request with a RUN command, or correct the program.

Example:

```
:10 FOR I=1 TO 20
:20 X=I*SIN(I*4)
:25 GO TO 100
:30 NEXT I: END
:100 PRINT "X=";X
:110 RETURN
:RUN
X=-.7568025

 110 RETURN
          ↑ERR 25
:25 GOSUB 100                              (Possible Correction)
```

Code 26

Error: ILLEGAL FOR/NEXT USAGE

Cause: There is no companion FOR statement for a NEXT statement, or a RERUN starting location is illegal.

Action: Repeat execution request with a RUN command, or correct the program.

Example:

```
:10 PRINT "I=";I
:20 NEXT I
:30 END
:RUN
I= 0
 20 NEXT I
        ↑ERR 26
:5  FOR I=1 TO 10                          (Possible Correction)
```

Code 27

Error: INSUFFICIENT DATA

Cause There is insufficient data for READ statement requirements.

Action: Correct program to supply additional data.

Example:

```
:10 DATA 2
:20 READ X,Y
:30 END
:RUN

 20 READ X,Y
           ↑ERR 27
:11 DATA 3                              (Possible Correction)
```

Code 28

Error: DATA REFERENCE BEYOND LIMITS

Cause: The data reference in a RESTORE statement is beyond the existing data limits.

Action: Correct the RESTORE statement.

Example:

```
:10 DATA 1,2,3
:20 READ X,Y,Z
:30 RESTORE 5
    ....
    ....
    ....
:90 END
:RUN
 30 RESTORE 5
            ↑ERR 28
:30 RESTORE 2                           (Possible Correction)
```

Code 29

Error: ILLEGAL DATA FORMAT

Cause: The data format for an INPUT statement is illegal (format error).

Action: Re-enter data in the correct format starting with erroneous number or terminate run with the ESCape (or ATTention) key and run again.

Example:

```
:10 INPUT X,Y
    ....
    ....
    ....
:90 END
:RUN
:INPUT
:1A,2E-30
  ↑ERR 29
:12,2E-30                               (Possible Correction)
```

Code 30

Error: EXPECTED LITERAL

Cause: An alphanumeric literal string (enclosed in quotes) was expected.

Action: Correct data value.

Example:
```
:10 READ A$
:20 DATA ABC
:RUN

 20 DATA ABC
          ↑ERR 30
:20 DATA "ABC"                    (Possible Correction)
```

Code 31

Error: UNDEFINED FN FUNCTION

Cause: An undefined FN function was referenced.

Action: Correct program to define or reference the function correctly.

Example:
```
:10 X=FNC(2)
:20 PRINT "X=";X
:30 END
:RUN

 10 X=FNC(2)
       ↑ERR 31
:05 DEF FNC(V)=COS(2*V)           (Possible Correction)
```

Code 32

Error: ILLEGAL FN USAGE

Cause: More than five levels of nesting were encountered when evaluating an FN function.

Action: Reduce the number of nested functions.

Example:
```
:10 DEF FN1(X)=1+X :DEF FN2(X)=1+FN1(X)
:20 DEF FN3(X)=1+FN2(X) :DEF FN4(X)=1+FN3(X)
:30 DEF FN5(X)=1+FN4(X) :DEF FN6(X)=1+FN5(X)
:40 PRINT FN6(2)
:RUN

10 DEF FN1(X)=1+X :DEF FN2(X)=1+FN1(X)
          ↑ERR 32
:40 PRINT 1+FN5(2)                (Possible Correction)
```

Code 33

Error: ILLEGAL VALUE FOR ARRAY DIMENSION

Cause: The values assigned for array dimensions exceed the allowable limits; a dimension is greater than 255; an array variable subscript exceeds the defined dimension; or illegal STR function specifications.

Action: Correct the program.

Example:

```
:10 DIM A(2,3)
:20 A(1,4)=1
:30 END
:RUN

 20 A(1,4)=1
          ↑ERR 33
:10 DIM A(2,4)                                   (Possible Correction)
```

Code 34

Error: EXPONENT OVERFLOW

Cause: The resulting magnitude of the number calculated in an arithmetic operation or entered was greater than or equal to 10^{63}.
(+, –, *, /, ↑, TAN, EXP, Matrix Operations).

Action: Correct the program in accordance with the limit.

Example:

```
:A=5E60*4E62
 A=5E60*4E62
            ↑ERR 34
```

Code 35

Error: EXPONENT UNDERFLOW

Cause: The resulting magnitude of the number calculated in an arithmetic operation or entered was less than or equal to 10^{-65}.
(+, –, *, /, ↑, TAN, EXP, Matrix Operations).

Action: Correct the program in accordance with the limit.

Example:

```
:PRINT (2E-65)/(2E+24)
 PRINT (2E-65)/(2E+24)
                      ↑ERR 35
```

Code 36

Error: DIVISION BY ZERO

Cause: A division by zero has been attempted.

Action: Correct the program data.

Example:

```
:25 X=0
:26 Y=Z/X

:RUN

26 Y=Z/X
        ↑ERR 36
```

Code 37

Error: ILLEGAL LOG FUNCTION ARGUMENT

Cause: A LOG function argument is zero or negative.

Action: Correct the program.

Example:

```
:PRINT LOG(-.8)

 PRINT LOG(-.8)
                 ↑ERR 37
```

Code 38

Error: ILLEGAL SQR FUNCTION

Cause: A SQR function argument is negative.

Action: Correct the program.

Example:

```
:PRINT SQR(-.8)

 PRINT SQR(-.8)
                 ↑ERR 38
```

Code 39

Error: INVALID EXPONENTIATION

Cause: An exponentiation, (X ↑ Y), was attempted where X was negative and Y was not integral, producing an imaginary result.

Action: Correct program data.

Example:

```
:50 D=(-3.2)↑ 2.5
:RUN

50 D=(-3.2) ↑ 2.5
                  ↑ERR 39
```

Code 40

Error: ILLEGAL SIN, COS, OR TAN ARGUMENT

Cause: The function argument exceeds 1E8.

Action: Change the function argument.

Example:

```
:10 X=SIN(6E12)
:RUN

 10 X=SIN(6E12)
                 ↑ERR 40
```

Code 41

Error: MATRIX NOT SQUARE

Cause: The dimensions of the operand in a MAT inversion or identity are not equal.

Action: Correct the array dimensions.

Example:

```
:10 MAT A=IDN(3,4)
:RUN

 10 MAT A=IDN(3,4)
                 ↑ERR 41
:10 MAT A=IDN(3,3)                    (Possible Correction)
```

Code 42

Error: SINGULAR MATRIX

Cause: The operand in a MAT inversion statement is singular and cannot be inverted.

Action: Correct the program.

Example:

```
:10 MAT A=ZER(3,3)
:20 MAT B=INV(A)
:RUN

 20 MAT B=INV(A)
              ↑ERR 42
```

Code 43

Error: ILLEGAL REDIMENSIONING OF ARRAY

Cause: The space required to redimension the array is greater than the space initially reserved for the array.

Action: Reserve more space for array in DIM or CON statement.

Example:

```
:10 DIM A(3,4)
:20 MAT A=CON(5,6)
:RUN

 20 MAT A=CON(5,6)
                ↑ERR 43
:10 DIM A(5,6)                        (Possible Correction)
```

Code 44

Error: ILLEGAL MATRIX OPERAND

Cause: The same array name appears on both sides of the equal sign in a MAT multiplication or transposition statement.

Action: Correct the statement.

Example:

```
:10 MAT A=A*B
          ↑ERR 44
:10 MAT C=A*B                         (Possible Correction)
```

Code 45

Error: MATRIX OPERANDS NOT COMPATIBLE

Cause: The dimensions of the operands in a MAT statement are not compatible; the operation cannot be performed.

Action: Correct the dimensions of the arrays.

Example:

```
:10 MAT A=CON(2,6)
:20 MAT B=IDN(2,2)
:30 MAT C=A+B
:RUN

 30 MAT C=A+B
          ↑ERR 45
:10 MAT A=CON(2,2)
```

(Possible Correction)

Code 46

Error: MISSING FILE VERB

Cause: A file verb was expected (FILEEND, FILESAVE, FILEREAD, etc.)

Action: Correct statement text.

Example:

```
:10 FILEREED #0,A,B
        ↑ERR 46
:10 FILEREAD #0,A,B
```

(Possible Correction)

Code 47

Error: ILLEGAL IMAGE STATEMENT

Cause: There are no "#" characters in the image statement or the total number of characters in a single format specification for numeric image exceeds 24 characters.

Action: Correct the image statement.

Example:

```
:10 % THE ANSWER IS - - -
                          ↑ERR 47
:10 % THE ANSWER IS ###
```

(Possible Correction)

Code 48

Error: ILLEGAL PRINT USING REFERENCE

Cause: A PRINT USING statement does not reference an image statement. (Statement with leading % character)

Action: Correct reference.

Example:

```
:10 GO TO 20
:20 PRINT USING 10,A
:30 % A=####
:RUN

10 GO TO 20
   ↑ERR 48
:20 PRINT USING 30,A
```

(Possible Correction)

Code 49

Error: ILLEGAL COMMON ASSIGNMENT

Cause: A COM statement variable definition was preceded by a non-common variable definition, or a RERUN was requested after new COM statements were added.

Action: Correct program, making all COM statements the first numbered lines, or execute with a RUN command.

Example:

```
:10 A=1 :B=2
:20 COM A,B
:99 END
:RUN

20 COM A,B
        ↑ERR 49
:10 (RET)
:30 A=1 :B=2
```

(Possible Correction)

PROGRAM LIMITS

An estimate of the partition size required to run a particular BASIC program is given by the following equation:

partition size = C + 5(N) + 18(A) + E

where:

C = number of characters in the program (including spaces, carriage, returns, etc.)

N = number of numeric variables (each element of an array is a variable)

A = number of alphanumeric variables (each element of an array is a variable)

E = space required for FOR and GOSUB processing and expression evaluation. E = 200 bytes for an average program but varies with program complexity.

APPENDIX A:

BASIC System Syntax Rules

(This page intentionally left blank)

APPENDIX A: BASIC System Syntax Rules

This appendix concisely states the syntax rules for the BASIC system. The following notation is used for the formal representation:

1. Square brackets ([]) indicate that the enclosed item is optional.
2. Braces ({}) enclosing vertically stacked items indicate that one of the items is required. Braces enclosing a group of items indicate the contents of the braces is required.
3. Ellipsis (...) indicates that the immediately preceding item may occur once or many times in succession.
4. Except within quotation marks ("), BASIC Syntax ignores blanks.

I. System Commands

A system command is a user request for the system to perform some global function. Any user input which is not recognized by the system as a system command is considered to be a program statement. All system commands must be terminated with a carriage return. System command syntax is:

LIST	['statement number' [,'statement number']]
LOAD	'unit' [/ 'integer']
RUN	
SAVE	'unit' [/ 'integer']
RERUN	['statement number']
START	
RESTART	

II. Program Statements.

There are two types of BASIC program statements: program statements and immediate statements. Program statements always begin with a statement number, while immediate statements do not. A list of program statement syntax is given below, followed by definitions of the individual syntax elements.

$$\text{COM} \left\{\begin{array}{l}\text{'scalar'}\\ \text{'array variable'}\end{array}\right\} \quad \left[, \left\{\begin{array}{l}\text{'scalar'}\\ \text{'array variable'}\end{array}\right\}\right] \ldots$$

$$\text{DATA} \left\{\begin{array}{l}\text{'number'}\\ \text{" 'text string' "}\end{array}\right\} \left[, \left\{\begin{array}{l}\text{'number'}\\ \text{" 'text string' "}\end{array}\right\}\right] \ldots$$

Note — " 'text string' " for Extended BASIC only.

$$\text{DEF FN} \left\{\begin{array}{l}\text{'letter'}\\ \text{'digit'}\end{array}\right\} \text{('scalar') = 'expr'}$$

DIM 'array variable' [{, 'array variable' }. . .]

END

FOR 'scalar' = 'expr' TO 'expr' [STEP 'expr']

GOSUB 'statement number'

GO TO 'statement number'

IF 'relation' THEN 'statement number'

INPUT 'variable' [{, 'variable' } . . .]

$$\text{[LET]} \left\{\begin{array}{l}\text{'numvar' [{ , 'numvar'} . . . .] = 'expr'}\\ \text{'alphavar' [{ , 'alphavar' } . . .] = } \left\{\begin{array}{l}\text{'alphavar'}\\ \text{" 'text string' "}\end{array}\right\}\end{array}\right\}$$

NEXT 'scalar'
PRINT [{ 't' 'print element' } . . .] ['t']
RANDOM
READ 'variable' [{, 'variable' } . . .]
REM 'text string'
RESTORE ['integer']
RETURN
STOP
TRACE $\left\{\begin{array}{l}\text{ON}\\\text{OFF}\end{array}\right\}$

III. Extended Basic Statements

CHAIN [R] 'unit'
FILEEND 'file number'
FILEMOD 'file number', $\left\{\begin{array}{c}\text{" 'text string' "}\\\text{SCRATCH}\end{array}\right\}$
FILEREAD 'file number', 'variable' [{ , 'variable' } . . .]
FILES 'unit' [/ 'integer'] [{ , 'unit' [/ 'integer'] } . . .]
FILESAVE 'file number'
FILEWRITE 'file number', 'pu element' [{ , 'pu element' } . . .]
GOTO 'statement number' [{ , 'statement number' } . . .] ON 'expr'
IF END 'file number' THEN 'statement number'

MAT 'letter' = 'letter' + 'letter'	(MAT addition)
MAT 'letter' = CON['redim expr']	(MAT CON function)
MAT 'letter' = 'letter'	(MAT equality)
MAT 'letter' = IDN['redim expr']	(MAT IDN function)
MAT 'letter' = INV('letter')	(MAT inversion)
MAT 'letter' = 'letter' * 'letter'	(MAT multiplication)
MAT 'letter' = ('expr') * 'letter'	(MAT scalar multiplication)
MAT 'letter' = 'letter' – 'letter'	(MAT subtraction)
MAT 'letter' = TRN('letter')	(MAT transposition)
MAT 'letter' = ZER['redim expr']	(MAT ZER function)

MAT FILEREAD 'file number', 'letter' [$] ['redim expr'] [{, 'letter' [$] ['redim expr'] } . . .]
MAT FILEWRITE 'file number', 'letter' [$] [{ , 'letter' [$] } . . .]
MAT INPUT 'letter' [$] ['redim expr'] [{ , 'letter' [$] ['redim expr'] } . . .]
MAT PRINT 'letter' [$] [{ 't' 'letter' [$] } . . .] ['t']
MAT PRINT USING 'statement number', 'letter' [$] [{ 't' 'letter' [$] } . . .] ['t']
MAT READ 'letter' [$] ['redim expr'] [{ , 'letter' [$] ['redim expr'] } . . .]
PRINT USING 'statement number', 'pu element' [{ 't' 'pu element' } . . .] ['t']
%" 'text string' " 'format specification' [{ " 'text string' " 'format specification' } . . .]

IV. Individual Syntax Elements

The table below lists alphabetically the definitions of all language syntax elements:

'alpha array element':: = 'letter' $ ('expr' [,'expr'])

'alphavar' :: = $\left\{\begin{array}{l}\text{'letter' \$}\\\text{'alpha array element'}\\\text{STR}\left(\left\{\begin{array}{l}\text{'letter' \$}\\\text{'alpha array element'}\end{array}\right\}\text{, 'integer' [, 'integer']}\right)\end{array}\right\}$

'array variable' :: = 'letter' ('integer' [,'integer'])
'builtin' :: = one of the three character text strings: SIN, COS, TAN, EXP, ATN, LOG, ABS, SQR, INT, RND, & SGN.
'digit' :: = digit 0 through 9

'exponent' :: = E $\left[\begin{Bmatrix}+\\-\end{Bmatrix}\right]$ 'integer' (integer less than 64)

'expr' :: = $\left[\begin{Bmatrix}+\\-\end{Bmatrix}\right]$ 'term'

'file number' :: = # 'integer'

'format specification' :: = [±] # [# ...] [.] [{ # } . . .] [!!!!]
where total length of specification is not greater than 24 characters.

'fraction' :: = . 'integer'

$$\text{'function' :: = }\left\{\begin{array}{l}\begin{Bmatrix}\text{FN 'letter'}\\\text{'builtin'}\end{Bmatrix}\text{ ('expr')}\\\text{BOOL ('relation')}\\\begin{Bmatrix}\text{AND}\\\text{OR}\end{Bmatrix}\text{ ('expr', 'expr', ...)}\end{array}\right\}$$

'integer' :: = 'digit' ['digit' . . .]

'letter' :: = any letter of the alphabet

'null' :: = no or blank characters

'num array element' :: = 'letter' (expr' [,'expr'])

$$\text{'number' :: = }\left\{\begin{array}{l}\text{'integer'}\\\text{'fraction'}\\\text{'integer' 'fraction'}\end{array}\right\}\quad\text{['exponent']}$$

$$\text{'numvar' :: = }\left\{\begin{array}{l}\text{'scalar'}\\\text{'num array element'}\end{array}\right\}$$

$$\text{'operand' :: = }\left\{\begin{array}{l}\text{'numvar'}\\\text{'number'}\\\text{'function'}\end{array}\right\}$$

$$\text{'print element' :: = }\left\{\begin{array}{l}\text{'variable'}\\\text{'expr'}\\\text{TAB ('expression')}\\\text{" 'text string' "}\\\text{'null'}\end{array}\right\}$$

$$\text{'pu element' :: = }\left\{\begin{array}{l}\text{'expr'}\\\text{" 'text string' "}\\\text{'variable'}\end{array}\right\}$$

'redim expr' :: = ('expr' [,'expr'])

$$\text{'relation' :: = 'expr'}\quad\begin{array}{c}\text{For}\\\underline{3310}\\\left\{\begin{array}{c}>\\<\\\neq\\=\\\leq\\\geq\end{array}\right\}\end{array}\quad\text{or}\quad\begin{array}{c}\text{For}\\\underline{3315}\\\left\{\begin{array}{c}>\\<\\<>\\=\\<=\\>=\end{array}\right\}\end{array}\quad\text{'expr'}$$

'scalar' :: = 'letter' ['digit']

$$\text{'standard character' :: = }\left\{\begin{array}{l}\text{'letter'}\\\text{'digit'}\end{array}\right\}$$

'statement number' :: = 'integer' (four digits or less)

$$\text{'t' :: = }\left\{\begin{array}{c}\text{,}\\\text{;}\end{array}\right\}$$

$$\text{'term' :: = }\left\{\begin{array}{l}\text{'operand'}\\\begin{array}{c}\text{('expr')}\\\text{or}\\\text{'operand'}\end{array}\quad\left\{\begin{array}{c}+\\-\\*\\/\\\uparrow\end{array}\right\}\quad\text{'term'}\end{array}\right\}$$

'text string' :: = any string of typewriter characters excluding carriage return, backspace, etc.

'unit' :: = $\left\{\begin{array}{l}\text{" 'text string' "}\\ \text{'integer' (1—32)}\\ \text{'null'}\end{array}\right\}$

'variable' :: = $\left\{\begin{array}{l}\text{'numvar'}\\ \text{'alphavar' (Extended BASIC only)}\end{array}\right\}$

APPENDIX B:

Sample Programs

(This page intentionally left blank)

B.1 Monthly Mortgage Payment Calculation

The following program illustrates the use of nested loops to perform a calculation a number of times with different values.

The problem is to calculate monthly mortgage payments for a \$50,000 loan with a \$10,000, \$15,000, and \$20,000 down-payment, at interest rates of 7.5%, 8.0%, 8.5%, and 9.0% per year, and for loan periods of 20, 25, and 30 years. The monthly payment is given by:

M=P*I/(1–(1+I)↑ (–N))

where:
P = principle
I = monthly rate of interest
N = number of monthly payments.

```
:05 REM––MONTHLY MORTGAGE PAYMENT
:10 PRINT "DWNPYMT", "INT. RATE", "YRS.", "MO. PYMT"
:20 FOR D = 10000 TO 20000 STEP 5000
:30 FOR R = .075 TO .09 STEP .005
:40 FOR Y = 20 TO 30 STEP 5
:50  P = 50000 – D
:60  I = R/12
:70  N = Y*12
:80  M = P*I/(1 –(1 + I) ↑ (–N))
:90  PRINT D, R, Y, INT (100*M)/100
:100 NEXT Y
:110 NEXT R
:120 NEXT D
:130 END
:RUN

DWNPYMT      INT. RATE          YRS.     MO. PYMT.
10000        7.5000000  E-02    20       322.23
10000        7.5000000  E-02    25       295.59
10000        7.5000000  E-02    30       279.68
10000        8.0000000  E-02    20       334.57
10000        8.0000000  E-02    25       308.72
10000        8.0000000  E-02    30       293.5
10000        8.5000000  E-02    20       347.12
10000        8.5000000  E-02    25       322.09
10000        8.5000000  E-02    30       307.56
10000        9.0000000  E-02    20       359.89
10000        9.0000000  E-02    25       335.67
10000        9.0000000  E-02    30       321.84
15000        7.5000000  E-02    20       281.95
15000        7.5000000  E-02    25       258.64
15000        7.5000000  E-02    30       244.72
15000        8.0000000  E-02    20       292.75
  .               .              .          .
  .               .              .          .
  .               .              .          .
```

B.2 Quadratic Equation Solution

This program computes the roots of second degree equations with real coefficients of the form:

A*X↑2 + B*X+C = 0

by the well-known formula:

X = (–B + SQR (B ↑ 2– 4*A*C))/(2*A)

When the discriminant M = B ↑ 2– 4*A*C < 0, the roots of the equation are complex; otherwise, the roots are real. Since we cannot compute the square root of a negative number, we compute SQR(ABS(M)) and then branch (line 40) into two cases: real roots or complex roots.

```
:05 REM––QUADRATIC EQUATION SOLUTION
:10 READ A, B, C
:20 M = B ↑ 2 – 4*A*C
:30 D = SQR (ABS(M))
:40 IF M < 0 THEN 70
:50 PRINT "REAL:", (–B+D)/(2*A), (–B–D)/(2*A)
:60 GO TO 10
:70 PRINT "COMPLEX:", –B/2*A, "+ or –", D/(2*A);"I"
:80 GO TO 10
:100 DATA 1, 7, 12
:110 DATA 1, 0, 2, 2, –4, –30
:999 END
:RUN
REAL:          –3       –4
COMPLEX:       –0       + or –    1.4142136 I
REAL:           5       –3
```

B.3 Change Maker

This program computes the change a customer will receive after paying for a bill. The change is broken down into the minimum number of bills and/or coins that the cashier must give to the customer. Note: here change can only be made with pennies, nickels, dimes, quarters, $1, $5, $10, and $20.

In the FOR,NEXT loop, it is first tested to see if the change, C, is greater than or equal to X which has as its first value $20. If it is not, the test is tried for the next smallest denomination, and then the next, etc. If C ⩾ X, then the maximum whole number of times, M, that X divides C is computed by line 150 and the resulting change is computed by line 160. The quantity and denomination are then printed out. The loop is then repeated for the next smallest denomination.

```
:05 REM--CHANGE MAKER
:10 PRINT "INPUT AMOUNT OF BILL"
:20 INPUT B
:30 PRINT "INPUT PAYMENT"
:40 INPUT P
:50 IF P≥ B THEN 100         (For 3310; for 3315, >=)
:60 PRINT "INSUFFICIENT PAYMENT"
:65 PRINT "INPUT ADDITIONAL PAYMENT"
:70 INPUT A
:80 P = P+A
:90 GO TO 50
:100 C = P -B
:110 PRINT "CHANGE IS:$", C
:120 IF C = 0 THEN 220
:130 PRINT "QUANTITY", "DENOMINATION"
:140 FOR I = 1 TO 8
:150 READ X
:160 IF X > C THEN 200
:170 M = INT(C/X)
:180 C=C- M*X
:190 PRINT M, X
:200 NEXT I
:210 DATA 20, 10, 5, 1, .25, .1, .05, .01
:220 END
:RUN
INPUT AMOUNT OF BILL
INPUT
:12.34
INPUT PAYMENT
INPUT
:13.00
CHANGE IS: $      .66
QUANTITY          DENOMINATION
2                 .25
1                 .1
1                 5.0000000  E-02
1                 1.0000000  E-02
```

B.4 Plotting

The following program illustrates the use of the TAB instruction in plotting functions. Here we plot f(x) = x ↑ 2 from −2 TO 2.

```
:05 REM––PLOT X ↑ 2
:10 FOR X = –2 TO 2 STEP .2
:20 PRINT X;
:25 PRINT TAB (16 + 5*X*X); "*"
:30 NEXT X
:40 END
:RUN

–2                                *
–1.8                           *
–1.6                        *
–1.4                      *
–1.2                     *
–1                     *
–.8                   *
–.6                  *
–.4                  *
–.2                  *
0                   *
.2                   *
.4                   *
.6                   *
.8                    *
1                      *
1.2                     *
1.4                       *
1.6                         *
1.8                           *
2                                 *
```

B.5 Mean, Variance, and Standard Deviation

Frequently, in statistical problems it is necessary to calculate the sums and sums of squares for sets of data. The technique used for the computer solution of such problems is illustrated in the following program.

The mean (M), variance (V), and standard deviation (D) of a set of data are given by the following formulae:

$$M = \sum_{i=1}^{n} X_i/N$$

$$V = (\sum_{i=1}^{n} X_i^2 - N * M^2)/(N-1)$$

$$D = \sqrt{V}$$

where N = number of data points and $X_1, X_2, \cdots, X_n$ are the data points. The number of data points and then the actual data points are entered for each set of data to be processed. The variables S and T sum the data and the squares of the data, respectively. Initially, these are set to zero. Then, a data point is read and added to S and the square of the number added to T. This is done for each data point by the FOR,NEXT loop. The mean and variance are then calculated and the desired results printed out. The program then returns to line 20 to process the next set of data.

```
:05 REM——MEAN, VARIANCE, ST. DEV.
:10 PRINT "MEAN", "VARIANCE", "ST. DEV."
:30 READ N
:40 S, T = 0
:50 FOR I = 1 TO N
:55 READ X
:60 S = S+X
:70 T = T+X↑2
:80 NEXT I
:90 M = S/N
:100 V = (T – N*M ↑ 2) / (N – 1)
:110 PRINT M, V, SQR (V)
:120 GO TO 30
:800 DATA 7, 1, 2, 3, 4, 5, 6, 7
:810 DATA 8, 5.4, 4.0, 6.2, 8.5, 6.2, 4.9, 7.8, 5
:999 END
:RUN
MEAN          VARIANCE          ST. DEV.
4             4.6666667         2.1602469
6             2.3057143         1.5184579
```

B.6 Matrix Multiplication

This program illustrates the use of the COM statement to segment programs that would be too large to fit into the computer memory in their entirety.

A matrix, M, is an array of numbers such as
$$\begin{array}{cccc} 1 & 2 & 3 & 7 \\ 4 & 5 & 6 & 3 \\ 1 & 2 & 9 & -1 \end{array}$$

The element in the "I" row and "J" column of M is denoted M_{ij}. The matrix is said to be of dimension NxM if N = number of rows and M = number of columns in M. The product of an NxM matrix A by an RxP matrix B is the NxP matrix C where:

$$C_{ij} = \sum_{k=1}^{M} a_{ik} * b_{kj}.$$ Note: it is required that R=M.

The first segment of this program reads the matrix elements. Matrix A is stored in the array variable A(I,J) such that A(I,J) = a_{ij}. Similarly, B(I,J) = b_{ij}. These array variables and the dimensions of the matrices are then saved in the memory for the second segment of the program and the rest of the first segment is cleared from the memory. The second segment performs actual multiplication and prints out the resulting matrix. Each element in the product C is computed by the FOR,NEXT loop in lines 70 – 90.

```
:05 REM--MATRIX MULTIPLICATION
:10 COM A (7,7), B(7,7), N, M, P, R
:20 READ N,M
:30 FOR I=1 TO N
:40 FOR J=1 TO M
:50 READ A(I,J)
:60 NEXT J
:70 NEXT I
:80 READ R,P
:90 IF R=M THEN 120
:100 PRINT "MATRICES INCOMPATIBLE"
:110 STOP
:120 FOR I=1 TO M
:130 FOR J=1 TO P
:140 READ B(I,J)
:150 NEXT J
:160 NEXT I
:800 DATA 4,3
:810 DATA 1, 0, -1
:811 DATA 2, 2, 2
:812 DATA -4, -4, 1
:813 DATA 3, 0, 0
:815 DATA 3, 4
:816 DATA 1, 2, 3, 4
```

B.6 Matrix Multiplication (continued)

```
:817 DATA –2, 0, 0, 1
:818 DATA 1, 1, 2, 2
:999 END
:RUN

END PROGRAM

:RESTART
:10 COM A(7, 7), B(7, 7), N, M, P, R
:20 DIM C(7,7)
:30 IF R=M THEN 40
:35 STOP
:40 FOR I=1 TO N
:50 FOR J=1 TO P
:60 C(I,J) = 0
:70 FOR K=1 TO M
:80 C(I,J) = C(I,J) + A(I,K) * B(K,J)
:90 NEXT K
:100 PRINT C(I,J),
:110 NEXT J
:130 NEXT I
:140 END

:RUN

0       1       1       2
0       6      10      14
5      –7     –10     –18
3       6       9      12
```

B.7 Simulation of Dice Game

This program illustrates how the computer can simulate real-life processes. In this case we simulate the game of "shooting craps". Lines 05 – 70 show how REM statements can be used to supply the user of a program with necessary information. In line 130, the player's bet is inputed. Then, in subroutine 300 – 340, the rolling of the dice is simulated using the function RND. An integer from 1 to 6 is generated for D1 (the value of the first die) and D2 (the value of the second die). Hence, D = D1 + D2 is the number rolled.

Next, in lines 140 – 180 it is determined whether the player wins, loses, or must roll again. If he loses (Rule 2), "YOU LOSE" is printed out (line 400); the total winnings are computed (line 410) and printed out (line 530). The program then returns to line 100 to take the next bet. Similarly, if the player wins (Rule 1), "YOU WIN" is printed out (line 500). The total winnings are computed (line 510) and printed out (line 530). The program returns to line 100 for next bet unless the house is broken in which case the program so states and ends. If the player must roll again, subroutine 300 – 340 is repeated and again it is determined whether the player wins, loses, or must roll again.

The program ends when the house is broken or a bet of $0 is inputed. This method of ending a program is commonly used when a program is to be repeated an arbitrary number of times.

```
1 RANDOM
05 REM——SIMULATION OF DICE GAME
10 REM——RULES
20 REM——1. IF 7 OR 11 IS ROLLED ON FIRST THROW,
25 REM——YOU WIN.
30 REM——2. IF 2, 3, OR 12 IS ROLLED ON FIRST THROW,
35 REM——YOU LOSE.
40 REM——3. IF ANOTHER NUMBER IS ROLLED, KEEP
50 REM——ROLLING UNTIL THIS NO. IS ROLLED AGAIN
60 REM——(YOU WIN) OR A 7 IS ROLLED (YOU LOSE).
70 REM——4. HOUSE LIMIT IS $1000.
80 A=0
90 PRINT
110 PRINT "YOUR BET IS?"
120 INPUT B
125 IF B=0 THEN 999
130 GOSUB 300
140 IF D<4 THEN 400
150 IF D=12 THEN 400
160 IF D=7 THEN 500
170 IF D=11 THEN 500
180 P=D
190 GOSUB 300
200 IF D=P THEN 500
210 IF D=7 THEN 400 :GOTO 190
300 REM——ROLL DICE
310 D1 = INT (6*RND(X)+1)
```

Simulation of Dice Game (continued)

```
320 D2 = INT(6*RND(X)+1)
330 D=D1+D2
333 PRINT "YOU ROLL:";D
335 PRINT D1,D2
340 RETURN
400 PRINT "YOU LOSE"
410 A=A–B
420 GOTO 530
500 PRINT "YOU WIN!"
510 A=A+B
530 PRINT "YOUR WINNINGS ARE $";A
535 IF A > 1000 THEN 900
540 GO TO 90
900 PRINT "CONGRATULATIONS!!!!  YOU BROKE THE HOUSE."
999 END
:RUN

YOUR BET IS?
INPUT
:100
YOU ROLL: 11
6                    5
YOU WIN!
YOUR WINNINGS ARE $ 100

YOUR BET IS?
INPUT
:250
YOU ROLL: 5
4                    1
YOU ROLL: 11
6                    5
YOU ROLL: 5
3                    2
YOU WIN!
YOUR WINNINGS ARE $ 350

YOUR BET IS?
INPUT
:200
YOU ROLL: 10
6                    4
YOU ROLL: 5
3                    2
YOU ROLL: 6
4                    2
YOU ROLL: 4
2                    2
YOU ROLL: 7
2                    5
YOU LOSE
YOUR WINNINGS ARE $ 150
```

WANG
LABORATORIES, INC.

836 North Street
Tewksbury, Massachusetts 01876
TELEPHONE (617) 851-7311
TWX 710-343-6769
TELEX 94-7421

WANG EUROPE, S.A.
Rue Anatole France 115-121
1030 Brussels, Belgium
TELEPHONE 15.48.01/02/03

WANG ELECTRONICS, LTD.
40-44 High Street
Northwood, Middlesex, England
TELEPHONE: Northwood 27677

WANG LABORATORIES GMBH
Moselstrasse No. 4
Frankfurt 6000
West Germany
TELEPHONE (611) 23.00.49

WANG LABORATORIES (CANADA) LTD.
180 Duncan Mill Road
Don Mills, Ontario
TELEPHONE (416) 449-7890
TELEX 06217549

WANG LABORATORIES (TAIWAN) LTD.
9, Sec. 5, Nanking East Road
Taipei, Taiwan
TELEPHONE 77.96.19 or 71.35.78

PHI COMPUTER SERVICES, INC.[*]
800 Massachusetts Avenue
Arlington, Massachusetts 02174
TELEPHONE (617) 648-8550

WANG MEDICAL SYSTEMS, INC.
836 North Street
Tewksbury, Massachusetts 01876
TELEPHONE (617) 851-7311
TWX 710-343-6769
TELEX 94-7421

Printed in U.S.A.
700-0260A
6-71-2M
Price $10.00