SDS

SCIENTIFIC DATA SYSTEMS

Reference Manual

SDS 940 BASIC

SUMMARY OF BASIC COMMANDS

	Command Syntax	Page
	step DATA constant-1[,constant-2] ... (RET)	9
	step DEF FNletter-1 (letter-2) = expression (RET)	14
*	DEL {step-1 {ALL / [-step-2] / [, step-2]...}} (RET)	17
	step DIM letter-1(expression-1[,expression-2])[,letter-2(expression-3[,expression-4])]... (RET)	11
*	DUMP [step-1 {[-step-2] / [, step-2]...}] (RET)	17
	step END (RET)	16
	step FOR variable = expression-1 TO expression-2 [STEP expression-3] (RET)	10
	step-1 GOSUB step-2 (RET)	15
**	[step-1] GO TO step-2 (RET)	8
	step-1 IF relational-expression THEN step-2 (RET)	8
	step INPUT[FILE] variable-1[,variable-2]... (RET)	9,12
**	[step] LET variable = expression (RET)	7
*	LIST [step-1 {[-step-2] / [, step-2]...}] (RET)	17
*	LOAD {TELETYPE / /file-name/} (RET)	10
	step NEXT variable (RET)	10
	step OPEN /file-name/, {INPUT / OUTPUT} (RET)	12
**	[step] PRINT {expression-1 / "character-string-1"} [{, / □ / ;} {expression-2 / "character-string-2"}]... (RET)	7,12
	step PRINT FILE expression-1[,expression-2]... (RET)	12
	step READ variable-1[,variable-2]... (RET)	9
	step REM character-string (RET)	16
	step RETURN (RET)	15
*	RUN (RET)	16
	step STOP (RET)	16

*System Directives
**Optionally, Statements or System Directives

Price: $1.00

BASIC

REFERENCE MANUAL

for

SDS 940 TIME-SHARING COMPUTER SYSTEMS

90 11 11B

January 1968

SDS

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

Printed in U.S.A.

REVISION

This publication, SDS No. 90 11 11B, is a minor revision of the SDS BASIC Reference Manual (dated March 1967). A change in the text is indicated by a vertical line in the margin of the page.

RELATED PUBLICATIONS

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

CONTENTS

ILLUSTRATIONS

1. INTRODUCTION

The BASIC[†] language and compiler were originally developed at Dartmouth College for time-sharing computer users with no previous knowledge of computers, as well as for users with considerable programming experience. Thus, BASIC is as useful to the businessman as it is to the scientist or engineer. A simple, straightforward language, BASIC closely resembles standard mathematical notation. In addition to its powerful arithmetic capability, it also contains editing features, simple input and output procedures, and complete language diagnostics.

This manual is intended to serve as a tutorial guide for the new BASIC user and as a reference source for the experienced . Material is presented in a sequence that will enable the reader to immediately use the SDS 940 time-sharing computer terminal to write simple programs, to gain confidence in the system, and then to progress to more difficult programs.

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

1. Underscored copy in an example represents copy produced by the system in control of the computer. Unless otherwise indicated, copy that is not underscored in an example must be typed by the user.
2. The (RET) notation appearing after some lines in the examples indicates a carriage return. The carriage return key is labeled RETURN on the Teletype keyboard. The user must depress the carriage return key after each command to inform the computer that the current command is terminated and a new one is to begin. The computer then upspaces the paper automatically.
3. Non-printing control characters are represented in this manual by an alphabetic character and a superscript c (e.g., D^C). The user depresses the alphabetic key and the Control (CTRL) key simultaneously to obtain a non-printing character. For editing purposes some control characters will cause a symbol to be printed, but this symbol does not appear in the final version of an edited line.
4. Other typographic conventions pertain to the method used in this manual to define the syntax of BASIC commands. These conventions are described in the paragraph titled "Basic Syntax Notation" later in this chapter.

OPERATING PROCEDURES

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

[†]The word "BASIC" is an acronym for "Beginner's All-purpose Symbolic Instruction Code".

LOG IN

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

1. If the FD-HD (Full Duplex-Half Duplex) switch is present, turn the switch to FD. This is a toggle switch with two locking positions. When the Teletype is not connected to the computer (sometimes called the Local Mode), this switch must be in the HD position. Whenever the Teletype is connected to the computer, this switch must be in the FD position.

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

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

   PLEASE LOG IN:

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

   PLEASE LOG IN: number password;name;project code

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

   PLEASE LOG IN: A1PASS;JONES;REPUB (RET)
   PLEASE LOG IN: B4WORD;BROWN;DEMO (RET)
   PLEASE LOG IN: C6PW;SMITH (RET)

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

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

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

   READY date, time

   -

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

6. In response to the dash, the user types

   BASIC ⏎

   The Executive will respond with the symbol >, at the beginning of the next line, which means that the BASIC subsystem is in control and waiting for a command.

ERROR CORRECTION

If the user makes a mistake while typing and notices it immediately, he can correct the error at once. The BASIC language will accept the following edit characters:

Character	Function
←	The left arrow (located on the letter "O" while the shift key is depressed) is used to delete the most recent character typed. If the user notices that he has just mistyped a letter or a symbol, he strikes the ← key, which tells BASIC to ignore the previous character. Striking this key repeatedly will delete a corresponding number of characters, but only to the start of the current line.

For example, the command

PRE ← INT A+C ← B ⏎

will appear to BASIC as

PRINT A+B ⏎

while

PRINT← ← ← ← ← ← ← LET X=Y ⏎

will appear to BASIC as

LET X=Y ⏎

A command preceded by a step number (see "Writing Programs in BASIC") may be deleted by retyping the step number immediately followed by a carriage return. A command preceded by a step number may be changed by retyping the step number and the new command, followed by a carriage return.

ESCAPE

The ESCAPE ⎋ key[†] may be used at almost any time. It causes the subsystem in control to abort the current operation and ask for a new command. Striking the ⎋ key before terminating a command with ⏎ aborts the command. BASIC responds by printing a > at the beginning of the next line.

EXIT AND CONTINUE

Striking the ⎋ key twice in succession causes computer control to return to the Executive. If the user wants to reenter BASIC without losing his program and if he has not called any other subsystem since leaving BASIC, he can type CONTINUE in response to the dash. This will return him to BASIC so that he can resume his work.

LOG OUT

When the user wishes to be disconnected from the computer, he types two consecutive escapes (to return control to the Executive) and then types

LOGOUT ⏎

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

BASIC ARITHMETIC COMPONENTS

The arithmetic components of the BASIC language are constants, variables and expressions. These components are described in the following paragraphs.

CONSTANTS

Constants may be expressed in BASIC in three ways. First, they may be expressed as integers (whole numbers, e.g., 1, 7, 130, 256). Constants may also be expressed in floating-point form; that is, as numbers with a decimal point (e.g., 1.5, 10.0, 155.55). Finally, constants may be written in scientific notation, which makes large numbers easier to express. For example, the number 53,000,000,000 may be written as 53E9. The letter E stands for "times ten to the power of"; thus, 53E9 means 53×10^9.

In all cases, constants in BASIC may be thought of as being in floating-point form. This allows the computer to perform addition, subtraction, multiplication and division automatically and makes it unnecessary for the user to worry about positioning the decimal point.

BASIC will accept and print a constant in integer form if the constant has 6 or fewer significant digits and if it has no fractional part. Otherwise, BASIC will round off the constant to the first seven significant digits and the constant will be expressed in scientific notation. The following examples illustrate these rules:

```
>PRINT 12345678912 ⏎
 1.234568E+10

>PRINT 123456789.0 ⏎
 1.234568E+08

>PRINT 123456789.23 ⏎
 1.234568E+08

>PRINT -12345.1E2 ⏎
-1.23451E+06

>PRINT 23456E10 ⏎
 2.3456E+14
```

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

VARIABLES

The numerical value of a constant is always the number itself. BASIC also permits the user to use a symbol to represent a number. Such symbols are called variables because the value of the symbol may be changed. Associated with each variable is a place in computer memory where the defined value of the variable is stored. In BASIC, a variable may be a single letter or a letter followed by a number. For example, some legal variables are

B
C
A1
Z5

and some illegal variables are

1B (first character is not a letter)
BA (second character is not a digit)
A35 (too many characters)

It is often convenient to keep data in a list. Such a list is called an array. The individual values in the list are called array elements. We refer to an array element by using the name of the array and the position of the element in the array. For example, we can refer to the fourth element in array A by writing A(4). In this example, the 4 is called the subscript.

When the array has only one "dimension", it is often called a vector or a linear array. However, arrays may have more than one dimension. A two-dimensional array, often called a matrix, may be thought of as having columns and rows. There is always one subscript for each dimension; thus, a two-dimensional array is written as A(X,Y)= where X represents a row number and Y represents the column number.

Note that the name of an array must be only one letter, while a subscript may be any evaluatable expression, which may also include an array element. (Expressions are discussed below.)

As a further example of matrix notation, consider the following chart, which lists the expenses for a four-day car trip:

	Column	1	2	3	4
Row	Item \ Date	June 5	June 6	June 7	June 8
1	Gasoline, oil	21.29	20.84	19.42	6.08
2	Tolls	1.32	.86	.40	.07
3	Food	11.18	12.82	14.39	5.06
4	Lodging	10.05	12.79	10.35	.00
5	Misc.	1.35	.90	.44	.10

If we consider the chart to be a two-dimensional array called P, the amount spent on June 5 for lodging would be represented by P(4,1). The first subscript always represents the row number, while the second subscript represents the column. Thus, the amount spent for food on June 8 would be represented by P(3,4).

EXPRESSIONS

Arithmetic Expressions

Arithmetic expressions are formed by combining variables and/or numbers with arithmetic operators, as in mathematical formulas. There are six arithmetic operators in BASIC:

- Negation (a unary operator)
↑ Exponentiation
* Multiplication
/ Division
+ Addition
- Subtraction

(↑, *, /, +, - : binary operators)

The following are examples of legal arithmetic expressions:

Expression	Meaning
F + H	F plus H
D + A + X	D plus A plus X
C - A(3)	C minus the third element of A
Q * -5	Q times minus 5
X/Y	X divided by Y
Q ↑ 2	Q to the power of 2 (or Q^2)
P ↑ R	P to the power of R (or P^R)
X/Y + 6	X divided by Y, plus 6

In the last example in the preceding table, the order of computation is not clear, i.e., the expression could have been interpreted as

X divided by the quantity Y + 6

or

the quantity X/Y plus 6

To make sure that the computer evaluates the expression the way the user meant it to be evaluated, there is an established rule of precedence:

> Exponentiation is always performed before negation and functions, which are always calculated before multiplication and division, which are always calculated before addition and subtraction. The computer calculates from left to right if operators of the same precedence (e.g., multiplication and division) appear in the same line.

To alter this order, parentheses must be used. Thus, to represent

X divided by the quantity Y + 6

we must write

X/(Y + 6)

Otherwise, according to the precedence rules it would be interpreted as "the quantity X/Y plus 6" because division is calculated before addition.

The following are examples of precedence in expressions:

Expression	Interpretation
A+B*C	A plus the quantity B times C.
(A+B)*C	C times the quantity A plus B.
Z-Y/X+W	Z minus the quantity Y divided by X, plus W.
(Z-Y)/(X+W)	the quantity Z minus Y divided by the quantity X plus W.
X ↑ 2+Y	X to the power of 2, plus Y
X ↑ -(2+Y)	X to the negative power of the quantity 2 plus Y.

Mathematical Functions

The mathematical functions available in BASIC are:

SIN(expression)	sine of expression in radians
COS(expression)	cosine of expression in radians
TAN(expression)	tangent of expression in radians
ATN(expression)	arc tan of expression in radians
EXP(expression)	natural exponent of expression
ABS(expression)	absolute value of expression
LOG(expression)	natural log of expression
SQR(expression)	square root of expression
LGT(expression)	log base 10 of expression
INT(expression)	integer part of expression
RND	random number

The expression enclosed in parentheses is called the argument. It may be any arithmetic expression, e.g., SQR(A*B). The arithmetic expression may include a function also, e.g., COS(N*X + SIN (T)) (a more complete discussion of functions is contained in Chapter 3).

Relational Expressions

A relational expression consists of two arithmetic expressions separated by one of the relational operators. The relational operators available in BASIC are

Operator	Relation
=	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
<>	not equal to

In BASIC, a relational expression is either "true" or "false", depending on whether the answer to the question implied by the relational expression is "yes" or "no", respectively.

For example, each of the following relational expressions imply a different question:

Relational expression	Question
X > 5	Is the value of the variable X greater than the constant 5?
A <> B	Are the values of the variables A and B unequal?
Z <= Y ↑ K	Is the value of the variable Z either less than or equal to the value of the variable Y raised to the power of K?

If the answer to the question is "yes", the relational expression is "true"; if the answer is "no", the relational expression is "false".

Evaluatable Expressions

An evaluatable expression is an arithmetic expression or a relational expression for which the values of all variables contained within the expression are known to BASIC at the time the command in which the expression appears is executed. In order for the value of a variable to be known, the variable must be defined (or "declared"). Variables may be declared by the execution of the following BASIC commands:

LET
READ and DATA (used together)
INPUT

These commands are described in Chapter 2.

BASIC SYNTAX NOTATION

The following paragraphs describe the notation used in this manual to define the syntax of the BASIC language.

1. A "notation variable" is used to represent a variable element of the BASIC language. A notation variable consists of lower-case letters, lower-case letters in combination with hyphens, or lower-case letters in combination with hyphens and digits (of which the first character is a letter). For example,

   variable

   denotes the occurrence of a BASIC variable.

2. A "notation constant" denotes the literal occurrence of the characters represented. A notation constant consists either of all capital letters or of a special character or symbol. For example,

   LET variable = expression (RET)

   denotes the literal occurrence of the word LET, followed by a BASIC variable, the literal occurrence of an equals sign, a BASIC arithmetic expression, and the literal occurrence of a carriage return.

3. The term "syntactical unit" (used in subsequent rules) is defined as a single notation variable or constant, or as any collection of notation variables, notation constants, BASIC language symbols, and reserved words surrounded by braces or brackets.

4. Braces { } are used to denote a grouping. For example,

$$\text{LOAD} \begin{Bmatrix} \text{TELETYPE} \\ \text{/file-name/} \end{Bmatrix}$$

The vertical stacking of syntactical units indicates that a choice is to be made. The above example indicates that the word LOAD must be followed by either the word TELETYPE or a slash character followed by a file name, followed by a second slash character.

5. Brackets [ ] denote options. Anything enclosed in brackets may appear one time or may not appear at all. For example,

[step-1] GO TO step-2 (RET)

denotes that the words GO TO may or may not be preceded by a step number; however, the words GO TO must be followed by a step number and a carriage return.

6. Elipsis marks . . . denote the occurrence of the immediately preceding syntactical unit one or more time in succession. For example,

expression-1 [,expression-2]. . . (RET)

denotes that the variable "expression-1" must occur; however, the variable "expression-2" may or may not occur since it is surrounded by brackets. If expression-2 does occur, it may be repeated one or more times (with a comma preceding each occurrence), each occurrence may have a unique form, and the last occurrence must be followed by a carriage return.

7. The character "i" is used as a collective reference designator when a syntactical unit may appear any number of times in succession. For example,

expression-i

denotes any of the expressions in the sample given above for rule 6.

BASIC NOTATION VARIABLES

The common variables used in this manual to define the syntax of the BASIC language are described in the following table.

Notation variable	Meaning
constant	a BASIC constant (see "Constants")
variable	a BASIC variable (see "Variables")
element	an element of an array (see "Variables") identified by its position in the array as in letter (expression-1 [,expression-2])
function	a mathematical function (see "Mathematical Functions") of the general form $\begin{Bmatrix} \text{SIN} \\ \text{COS} \\ \text{TAN} \\ \text{ATN} \\ \text{EXP} \\ \text{ABS} \\ \text{LOG} \\ \text{SQR} \\ \text{LGT} \\ \text{INT} \\ \text{RND} \end{Bmatrix}$ (expression)
expression	an arithmetic expression (see "Arithmetic Expressions") of the general form $\begin{Bmatrix} \text{constant-1} \\ \text{variable-1} \\ \text{element-1} \\ \text{function-1} \end{Bmatrix} \left[\begin{Bmatrix} + \\ - \\ * \\ / \\ \uparrow \end{Bmatrix} \begin{Bmatrix} \text{constant-2} \\ \text{variable-2} \\ \text{element-2} \\ \text{function-2} \end{Bmatrix} \right] \ldots$
relational-expression	a relational expression (see "Relational Expressions") of the general form expression-1 $\begin{Bmatrix} = \\ > \\ >= \\ < \\ <= \\ <> \end{Bmatrix}$ expression-2

BASIC NOTATION CONSTANTS

The notation constants of the BASIC language consist of command keywords, function identifiers, operators, and delimiters.

Command Keywords

The command keywords in the BASIC language are

Keyword	Use
DATA	defines data for a READ command
DEF	defines a nonstandard function
DEL	deletes a portion of a program
DIM	defines the dimensions of an array
DUMP	designates portion of a program to be saved
END	identifies the end of a program
FILE	specifies a system file
FN	used with DEF
FOR	part of FOR command
GO	part of GO TO command
GOSUB	calls a subroutine
IF	conditional GO TO command
INPUT	specifies Teletype or input file with OPEN
LET	arithmetic assignment

Keyword	Use
LIST	lists portion of program
LOAD	specifies program loading
NEXT	terminates FOR command group
OPEN	opens file for input or output
OUTPUT	specifies output file with OPEN command
PRINT	prints data on teletype or on file
READ	accepts data from DATA command
REM	inserts documentary comments
RETURN	terminates subprogram
RUN	begins execution
STEP	part of FOR command
STOP	terminates program execution
TELETYPE	designates remote terminal
THEN	part of IF command
TO	part of FOR and GO TO commands

Function Identifiers

The following function identifiers are a standard part of the BASIC language.

ABS	ATN	COS	EXP	INT	
LGT	LOG	RND	SIN	SQR	TAN

Other function identifiers may be declared by the programmer, as described in Chapter 3.

Operators

The operators included in the BASIC language are the arithmetic and relational operators. (See "Arithmetic Expressions" and "Relational Expressions".) Note that the equal to (=) operator is a replacement operator (instead of a relational operator) in the LET, FOR, and DEF commands.

Delimiters

Certain special characters are used as separators and as other types of delimiters. The BASIC delimiters are

Character	Use
,	separates elements of a list
;	separates elements of a formatted line
( )	encloses array subscripts and controls the evaluation of expressions
(RET)	terminates a command
"	denotes the beginning or the end of a text character string
blank	ignored (except in text)

It is important to note that blanks are ignored when they appear within a keyword, a constant, or a variable name. (For the purpose of this discussion, the symbol for a blank is □). The following example demonstrates the general rules.

35□5□PR□INT□"□SAMPLE□TEXT□"□B□8

ignored blanks — these blanks are part of text — ignored blanks

In the above example, "B" and "8" are interpreted as the variable "B8".

Note that blanks may be eliminated entirely if the programmer is not concerned with the readability of the printed copy. Furthermore, since each BASIC command is limited to a single line of 80 characters, the completion of a complex command may be more important than its readability.

2. BASIC COMMANDS

SINGLE COMMANDS IN BASIC

The BASIC language allows the remote Teletype terminal to be used as an extremely powerful desk calculator to evaluate complicated mathematical expressions and to reduce data that requires involved calculations. Two commands enable the user to express almost any mathematical expression – PRINT and LET.

PRINT

The PRINT command causes BASIC to print the value of an expression. The user types the word PRINT and the expression he wishes to evaluate (followed by a carriage return). The computer will then issue a line feed and print the value of the expression. The format of the PRINT command used for this operation is

PRINT expression (RET)

Example:

```
>PRINT 7.56*8.73 (RET)
 65.9988
```

Note that underlined copy is that which is generated by the computer. Several expressions can be evaluated with one PRINT command by separating the expression with commas. The format of the PRINT command used for successive evaluation and printing is

PRINT expression-1 [, expression-2]. . . (RET)

Example:

```
>PRINT 5*6, 7+8+40, 45/9 (RET)
 30          55          5
```

Text can also be printed by enclosing the characters to be printed in quotation marks according to the format

PRINT "character-string" (RET)

BASIC will cause the computer to print exactly what appears between the quotation marks, e.g.,

```
>PRINT "THIS MESSAGE INCLUDES BLANKS" (RET)
THIS MESSAGE INCLUDES BLANKS
```

the PRINT command can also be used to type text and values of expressions, e.g.,

```
>PRINT "A = " 5+6 (RET)
A = 11
```

More elaborate output formats can be constructed as described in Chapter 4.

LET or REPLACEMENT

If a variable is used as part of an expression in the PRINT command, the user must first assign a value to the variable. The LET command is used in other BASIC systems (notably, the Dartmouth BASIC compiler) to assign the value of an expression to a variable. In 940 BASIC, LET is not required but is accepted to preserve compatibility with these systems.

The format of the replacement statement which assigns the value of an expression to a variable, is

variable = expression (RET)

or

LET variable = expression (RET)

For example, the command

X = 2+3 (RET)

assigns the value 5 to the variable X; the command

Y = 10 (RET)

assigns the value 10 to the variable Y; and the command

A(5) = 9 (RET)

assigns the value 9 to the fifth element of array A.

Once the variable has a value, it may be used in a PRINT command, e.g.,

```
>A = 5 (RET)
>PRINT A (RET)
 5
>B = -4 (RET)
>PRINT B (RET)
-4
>PRINT "A + B =" A + B (RET)
A + B = 1
```

A replacement statement can be used to change the value of a variable at any time, e.g.,

```
>A = 5 (RET)
>PRINT "A =" A (RET)
A = 5
>A = A + 1 (RET)
>PRINT "NOW A =" A (RET)
NOW A = 6
```

WRITING PROGRAMS IN BASIC

The foregoing concerns some of the things BASIC can do when commands are entered and executed one at a time. This discussion of BASIC continues by introducing a second way of using it, that is, as a means for writing and storing computer programs for future execution.

A program is composed of commands (steps) that are to be used in solving a problem. For example, consider the following steps of a program that calculates the hypotenuse of a right triangle according to the formula:

$$\text{HYPOTENUSE} = \sqrt{(\text{side 1})^2 + (\text{side 2})^2}$$

```
>100 A = 4 (RET)
>110 B = 3 (RET)
>120 C = SQR(A ↑ 2 + B ↑ 2) (RET)
>130 PRINT "A=" A, "B=" B, "C=" C (RET)
>RUN (RET)
```

Note that each step has a unique step number (which may be any integer in the range 1 through 99999). The presence of the step numbers tells BASIC that these steps are not to be immediately executed, but are to make up a program. When the RUN command is given, telling the computer to execute the program, the steps are executed one at a time in ascending numerical sequence by step number. The result printed by the computer would be:

A= 4 B= 3 C= 5

Remember that the = sign in the program means replacement, not equality. Thus, step 100 means "assign the value 4 to A".

The following program calculates and prints the area and volume of a sphere:

```
>200 P = 3.14 (RET)
>300 R = 2 (RET)
>400 A = 4 * P * R ↑ 2 (RET)
>500 V = (4/3) * P * R ↑ 3 (RET)
>600 PRINT R, A, V (RET)
>RUN (RET)
```

When the RUN command is given, BASIC will print:

2 50.24 33.49333

Although BASIC executes commands according to the numerical sequence of step numbers, the steps of a BASIC program need not be prepared in numerical sequence. For example, the above program could have been prepared as

```
>200 P = 3.14 (RET)
>300 A = 4 * P * R ↑ 2 (RET)
>400 V = (4/3) * P * R ↑ 3 (RET)
>500 PRINT R, A, V (RET)
>250 R = 2 (RET)
>RUN (RET)
```

and the results will be identical.

GO TO

As we have seen, BASIC executes the steps of a program in ascending numerical sequence by step number. However, in writing programs it is sometimes necessary to change the normal sequence of execution. This can be accomplished by using the GO TO command, which has the format

[step-1] GO TO step-2 (RET)

where

step-1 is the optional step number of the GO TO command. If step-1 is not specified, BASIC begins executing commands (beginning with step-2) immediately after the carriage return.

step-2 is the step number of the command that is to be executed next (instead of the command with the next step number that is numerically higher than step-1).

Examples:

```
GO TO 100 (RET)
385 GO TO 215 (RET)
```

IF

It is often convenient to GO TO a step only under certain conditions. This type of statement is called the IF (or conditional GO TO) command, which has the format

step-1 IF relational-expression THEN step-2 (RET)

This command means, "If the relational expression is true, go to step-2 for the next command; otherwise (i.e., if the relational expression is false) go to the next step number in numerical sequence after step-1". For example, if we want to say, "If X is greater than 5, go to Step 100", we would write

```
70 IF X > 5 THEN 100 (RET)
```

Some other examples are

```
100  IF A = 10 THEN 500 (RET)
500  IF C(5) > 10 THEN 2300 (RET)
2300 IF D < = E THEN 100 (RET)
```

The following program solves a quadratic equation of the form $ax^2 + bx + c = 0$, by using the formulas:

$$X_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$X_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

```
>100 A = 5 (RET)
>200 B = 6 (RET)
>300 C = 7 (RET)
>350 D = B ↑ 2 - 4 * A * C (RET)
>400 IF D < 0 then 700 (RET)
>450 X1 = (-B + SQR(D)) / (2 * A) (RET)
>500 X2 = (-B - SQR(D)) / (2 * A) (RET)
>550 PRINT X1, X2 (RET)
>600 GO TO 750 (RET)
>700 PRINT "NO REAL ROOTS" (RET)
>750 STOP (RET)
>GO TO 100 (RET)
```

Execution of the above program begins immediately after the carriage return following the second GO TO command. Note that in the example, the second GO TO command could have been used to start execution at any step, whereas the RUN command always causes execution to begin with the lowest-numbered step. Note also that step 400 is a conditional GO TO command. It tells BASIC to skip the intervening steps and execute step 700 only if the discriminant (D) is less than zero. If the discriminant is not less than zero the next command (step 450) is executed after step 400.

DATA/READ

Values can be assigned to variables in several ways. The LET statement is one method. Another method involves the combined use of the DATA and READ commands.

All the constants that are to be assigned to variables throughout the program are written together in a DATA command, which has the format

step DATA constant-1[, constant-2]. . . (RET)

Examples:

125 DATA 5, 10, 15 (RET)
150 DATA 100, 0, 4E2, 4.2 (RET)
345 DATA 1.1, 1.7, 34902, 33.367E-15 (RET)

Similarly, the READ command has the format

step READ variable-1[, variable-2]. . . (RET)

Examples:

40 READ D (RET)
50 READ A1, A2 (RET)
60 READ X(1), X(2), X(3) (RET)

Each time a READ statement appears, the computer automatically assigns each constant-i in the DATA list to the corresponding variable-i in that READ statement. For example, the commands

100 READ A, B, C (RET)
200 DATA 1, 2, 3 (RET)

would be equivalent to the commands

100 A = 1 (RET)
150 B = 2 (RET)
200 C = 3 (RET)

Generally a program uses more than one value for a variable in order to prevent excessive use of constants and assignments. For example, consider the program

>10 G = 100 (RET)
>20 P = 20 (RET)
>30 D = G * P * .01 (RET)
>40 A = G - D (RET)
>50 PRINT D, A (RET)
>55 G = 150 (RET)
>60 P = 5 (RET)
>70 D = G * P * .01 (RET)
>80 A = G - D (RET)
>90 PRINT D, A (RET)

Another way of writing this program using the READ and DATA commands is

> 10 READ G, P (RET)
> 30 D = G * P * .01 (RET)
> 40 A = G - D (RET)
> 50 PRINT D, A (RET)
> 60 READ G, P (RET)
> 70 D = G * P * .01 (RET)
> 80 A = G - D (RET)
> 90 PRINT D, A (RET)
>100 DATA 100, 20, 150, 5 (RET)

Note that all the data that is to be assigned to G and P is now located in step 100.

INPUT

There is a third method of assigning values to variables which may be done at execution time, contrary to the other methods mentioned. To use the READ and DATA commands or the LET command, the user must assign values to all variables when the program is written. To assign values to variables at execution time the user may use the INPUT command, which has the format

step INPUT variable-1[, variable-2]. . . (RET)

Examples:

400 INPUT A, B, C, D, E (RET)
500 INPUT X, Y, Z1, Z2 (RET)

Each time the input command is encountered during execution, the program is halted and a ? is output to the terminal. At that time, numbers corresponding to the variable list, separated by commas, must be typed in, terminating with a carriage return. The values are automatically assigned to the respective variables and execution is continued.

Now the sample program given for the READ and DATA commands could be written as:

>10 INPUT G, P (RET)
>20 D = G * P * .01 (RET)
>30 A = G - D (RET)
>40 PRINT D, A (RET)
>50 INPUT G, P (RET)
>60 D = G * P * .01 (RET)
>70 A = G - D (RET)
>80 PRINT D, A (RET)
>RUN (RET)

When the RUN command is given, BASIC executes the first INPUT command and waits for the values of G and P, and a carriage return. Upon receiving these, execution continues until the next input command. Output would appear as

? 100, 20 (RET)
20 80
? 150, 5 (RET)
7.5 142.5

WRITING PROGRAM LOOPS

Note that the first four steps of the sample program for the INPUT command are exactly like the second four steps. This makes it possible to represent the program in the following way:

>10 READ G, P (RET)
>20 D = G * P * .01 (RET)
>30 A = G - D (RET)
>40 PRINT D, A (RET)
>50 GO TO 10 (RET)
>60 DATA 100, 20, 150, 5 (RET)
>RUN (RET)

The computer will perform steps 10 through 50 in the normal fashion, but after completing step 50 it will go back to step 10 and repeat steps 10 through 50. When the computer comes to step 50 once more, step 50 sends it back to step 10 again. This process is repeated over and over until all data defined in the DATA command (or any higher numbered DATA command) has been used. At this time, the message "OUT OF DATA 10" would be typed. This technique (often called a loop) is perhaps the single most important feature in programming. The following example shows all the steps that are necessary to set up a controlled loop to print all the numbers between 1 and 100.

```
>10 I = 1 (RET)
>15 IF I < 100 THEN 60 (RET)
>20 PRINT I (RET)
>45 I = I + 1 (RET)
>50 GO TO 15 (RET)
>60 PRINT "FINISHED" (RET)
>RUN (RET)
```

First, a variable was selected to be the counter (I). Second, an initial value was assigned to the counter variable (1). Third, the value of the counter variable was tested to see if it was finished (I > 100). Fourth, the value of the counter variable was increased each time the loop was repeated (I = I + 1).

FOR/NEXT

A second (and briefer) method of constructing program loops is to use the FOR and NEXT commands. The FOR command, which has the format

step FOR variable = expression-1 TO expression-2 (RET)

assigns the value of expression-1 to the variable and uses expression-2 as an upper limit for the value of the variable. The NEXT command, which has the format

step NEXT variable (RET)

must appear somewhere after the FOR command. The variable must be the same variable given in the FOR command. The purpose of NEXT is to increment the value of the variable (by 1) and compare its incremented value with the value of expression-2 given in the previous FOR command. If the incremented variable is less than or equal to expression-2, BASIC interprets the NEXT command as, "GO TO the previous FOR command". However, if the incremented value of the variable is greater than the value of expression-2 given in the FOR command, BASIC interprets the NEXT command as, "GO TO the next step in numerical sequence after the NEXT command".

The FOR loop is always executed at least once, even when expression-1 is initially greater than expression-2.

Thus, using the FOR and NEXT commands, the program given above could also be written as:

```
>10 FOR I = 1 TO 100 (RET)
>20 PRINT I (RET)
>30 NEXT I (RET)
>40 PRINT "FINISHED" (RET)
>RUN (RET)
```

Exactly the same looping procedure is followed; however, it happens automatically by using the FOR command, where

I is the counter variable
1 is the initial value of I
I > 100 is the test for completion
1 is the increment to be added to I.

In this sample program the "body" of the loop consists of one step (step 20). The body of the loop may be any number of steps, but it is always terminated by the NEXT command. When the loop is finished the next step executed is the step following the NEXT command.

In some program loops it is necessary to increment the counter variable by a value greater than 1. This can be accomplished by inserting a LET command immediately before the NEXT command. For example, to find and print all even numbers in the range 50 through 76, the following program could be used:

```
>10 FOR X = 50 TO 76 (RET)
>20 PRINT X (RET)
>25 X = X + 1 (RET)
>30 NEXT X (RET)
>40 PRINT "FINISHED" (RET)
>RUN (RET)
```

A simpler way of doing this is to use a STEP clause in the FOR command, as in the format

step FOR variable = expression -1 TO expression-2 [STEP expression-3] (RET)

where expression-3 is the increment to be added to the variable when the NEXT command is executed. Thus, the above program can also be written as

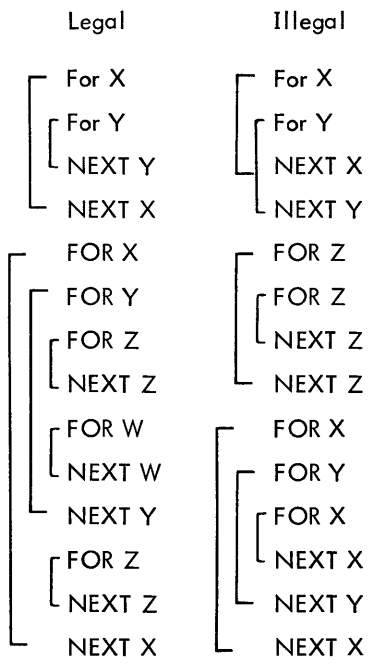
```
>10 FOR X = 50 TO 76 STEP 2 (RET)
>20 PRINT X (RET)
>30 NEXT X (RET)
>40 PRINT "FINISHED" (RET)
>RUN (RET)
```

According to the looping procedure

X is the counter variable
50 is the initial value of X
X > 76 is the test for completion
2 is the increment to be added to X.

Note that the increment is assumed to be 1 unless a STEP clause is added to the FOR command.

It is often useful to have loops within loops. These "nested" loops can be expressed with FOR and NEXT commands. In the following skeleton examples, the enlarged brackets mark the body of the loop.



USE OF SUBSCRIPTS

The concept of subscripting and arrays becomes extremely useful in relation to programming loops. Consider the following table, which lists the quantity of each type of item sold by each of five salesmen in one week.

	Salesman				
	Jones	Smith	Brown	Doe	White
Item 1	40	20	37	29	42
Item 2	10	16	3	21	8
Item 3	35	47	29	16	33

The price of each item is listed in the following table:

Item	Price
1	$1.25
2	$4.30
3	$2.50

In the following discussion, the quantities of items in the first table are regarded as the two-dimensional array Q(I, S) where I is the item number (row reference) and S is the salesman (column reference). The prices of the items are regarded as the one-dimensional array P(I) where I is the item number.

The following program calculates the total sales in dollars for each salesman using data from the preceding tables:

```
>10 FOR I = 1 TO 3 (RET)
>20 READ P(I) (RET)
>30 NEXT I (RET)
>40 FOR I = 1 TO 3 (RET)
>50 FOR S = 1 TO 5 (RET)
>60 READ Q(I, S) (RET)
>70 NEXT S (RET)
>80 NEXT I (RET)
>90 FOR S = 1 TO 5 (RET)
>100 T = 0 (RET)
>110 FOR I = 1 TO 3 (RET)
>120 T = T + P(I) * Q(I, S) (RET)
>130 NEXT I (RET)
>140 PRINT "TOTAL SALES FOR SALESMAN" S,"$"T (RET)
>150 NEXT S (RET)
>200 DATA 1.25, 4.30, 2.50 (RET)
>210 DATA 40, 20, 37, 29, 42 (RET)
>220 DATA 10, 16, 3, 21, 8 (RET)
>230 DATA 35, 47, 29, 16, 33 (RET)
>RUN (RET)
```

Steps 10 through 30 read in the values of the list P. Steps 40 through 80 read in the values the table Q. Steps 90 through 150 compute T (the total sales for each of the five salesmen) and print each answer as it is computed. The calculation for a single salesman takes place in steps 100 through 130. In steps 90 through 150, the letter I stands for the item and the letter S stands for the salesman.

DIM

BASIC automatically provides 11 elements (locations in memory) for a one-dimensional array, so that the subscript may vary from 0 to 10. If the user wants to allow for a longer array, he may specify its subscript range with the DIM (dimension) command, which has the format

step DIM array-1[, array-2]. . . (RET)

where each array-i has the form

letter-i (expression-i)

Each expression-i must evaluate as zero or a positive integer value, which specifies the upper subscript limit of the array-i as the value of the expression-i. (The lower subscript limit is 0).

Examples:

```
10 DIM Q(59) (RET)      declares an array of 60 elements
20 DIM B(A + X), Z(A(X)) (RET)
```

Similarly, the subscripts of a two-dimensional array may each vary from 0 to 10. This allows for 11 x 11 or 121 elements. If a larger table is desired, the user may use the DIM command with each array-i having the form

letter-i (expression-1, expression-2)

Each expression-i must evaluate as zero or a positive integer, which specifies the upper subscript limit for the corresponding dimension of the array-i.

Examples:

```
1 DIM Y(40, 50) (RET) declares an array of 41 x 51 elements
2 DIM X(Y, Z), A(1, 1) (RET)
3 DIM A(X(1), X(2)) (RET)
4 DIM L(M + N * N) (RET)
```

INPUT/OUTPUT COMMANDS

All data permanently stored at the computer installation is kept in files on a large disc memory system. Each user has his own files and file directory (for further description of files see the SDS 940 Terminal User's Guide).

OPEN

Often the values to be assigned to program variables are located in a permanent file. To read a permanent file, it must be in the user's file directory and it must be opened using the OPEN command. Similarly, program output may be written on a file rather than on the Teletype. Again, the file must appear in the user's file directory and must be opened using the OPEN command. The OPEN command has the format

step OPEN/file-name/, {INPUT / OUTPUT} (RET)

where file-name is the name of a file in the user's file directory.

Examples:

```
10 OPEN/DATA1/, INPUT (RET)
20 OPEN/XYZ/, INPUT (RET)
30 OPEN/MASTER/, OUTPUT (RET)
40 OPEN/F1/, OUTPUT (RET)
```

The OPEN command resets the file's location counter to its beginning so that the first value subsequently input from or output to the file will be the first value of the file. A file cannot be open for input and output simultaneously.

INPUT FILE

Once the file is opened for input, the user may read it by using the INPUT FILE command, which has the format

step INPUT FILE variable-1[, variable-2]. . . (RET)

Each time the INPUT FILE command is encountered during execution, the next value-i appearing on the file most recently opened for input is read and assigned to the next variable-i in the list of variables.

Examples:

```
50 INPUT FILE X (RET)
60 INPUT FILE A1, B, Z (RET)
```

The following technique can be used to open a file for INPUT (OUTPUT) which had been opened for OUTPUT (INPUT).

```
 5 OPEN /A/, INPUT (RET)          Opens /A/ for input
10 INPUT FILE X, Y, Z (RET)       Inputs data from /A/
15 OPEN /B/, INPUT (RET)          Closes /A/, Open /B/
                                  for input
20 INPUT FILE Q, R, S (RET)       Inputs data from /B/
25 OPEN /C/, INPUT (RET)          Closes /B/, Open /C/
                                  (a dummy) for input
30 OPEN /B/, OUTPUT (RET)
35 PRINT FILE G, H, I, J (RET)    Outputs data to file /B/
```

PRINT FILE

Once the file is opened for output, the user may write on it using the PRINT FILE command, which has the format

step PRINT FILE expression-1[, expression-2]. . . (RET)

Each time the PRINT FILE command is encountered during execution, the value of each expression-i appearing in the list of expressions is appended to the file most recently opened for output in the same order as given in the expression list.

Examples:

```
70 PRINT FILE X, Y (RET)
80 PRINT FILE X + Z/100, I (RET)
```

The following small programs illustrate one method of utilizing data files. Briefly, the first program creates a master file consisting of an employee number, hourly rate, and the number of hours worked for each company employee.

The second program reads the master file, calculates and prints the weekly salary for each employee (time and a half for overtime), and calculates total payroll.

The third program reads the master file and calculates company overtime.

```
>110 OPEN/MASTER/, OUTPUT (RET)
>111 INPUT E, R, H (RET)
>112 PRINT FILE E, R, H (RET)
>113 IF E = 0 THEN 110 (RET)
>114 END (RET)

>209 S = 0 (RET)
>210 OPEN/MASTER/, INPUT (RET)
>211 INPUT FILE E, R, H (RET)
>212 IF E = 0 THEN 220 (RET)
>213 IF H > 40 THEN 218 (RET)
>214 P = R * H (RET)
>215 PRINT E, P (RET)
>216 S = S + P (RET)
>217 GO TO 211 (RET)
>218 P = 40 * R + (H-40) * R * 1.5 (RET)
>219 GO TO 215 (RET)
>220 PRINT "TOTAL PAYROLL", S (RET)
>221 END (RET)

>309 S = 0 (RET)
>310 OPEN/MASTER/, INPUT (RET)
>311 INPUT FILE E, R, H (RET)
>312 IF E = 0 THEN 315 (RET)
>313 IF H < 40 THEN 311 (RET)
>314 S = S + (H-40) (RET)
>315 GO TO 311 (RET)
>316 PRINT "COMPANY OVERTIME", S (RET)
>317 END (RET)
```

PRINT

The PRINT command may be used simply and directly, as previously described, or it may be used in conjunction with output format options for the programmer who wants formatted output. For this purpose, there are three format types: zoned, packed, and compressed.

Zoned Format

The Teletype line is divided by BASIC into five zones of fifteen spaces each, which allows for the printing of up to five items per line. A comma is a signal to BASIC to move to the next print zone, or to the first print zone of the next line if it has just filled the fifth print zone. The termination of a PRINT statement signals a new line (unless a comma is the last symbol). Each number occupies one zone, whereas text occupies an integer number of zones; i.e., if text occupies part of a zone, the rest of the zone is filled with blanks. If text runs through the fifth zone, part of it may be lost.

Packed Format

The user may specify that output is to be printed in packed format by separating the expressions with the semicolon instead of the comma. Whereas the comma causes BASIC to move to the next zone to print the next item, the semicolon causes BASIC to move to the beginning of the next multiple of three characters to print the next answer. Thus, with packed output, the user can print eleven, three-digit numbers per line, eight, six-digit numbers per line or six, nine-digit numbers per line.

Compressed Format

If neither a comma nor a semicolon appears between expressions to be printed, the values will be printed without any separating spaces. Also, if two parts of text are separated only by the usual quotes, they will be printed without spaces. Similarly, if text is followed by an expression, without a comma or semicolon, the value will be printed immediately following the text (a blank replaces the plus sign for non-negative numbers).

Examples:

```
>1 FOR X = 1 TO 3 (RET)
>2 PRINT "X =" X, (RET)                    (zoned format)
>3 NEXT X (RET)
>4 END (RET)
>RUN (RET)
X = 1           X = 2           X = 3
```

```
>2 PRINT "X =" X; (RET)                    (packed format)
>RUN (RET)
X = 1  X = 2  X = 3
```

```
>2 PRINT "X =" X (RET)                     (compressed format)
>RUN (RET)
X = 1
X = 2
X = 3
```

A more extensive demonstration of packed format is shown in Figure 1.

```
>100 FOR X = 100 TO 125 (RET)
>110 PRINT X; (RET)
>120 NEXT X (RET)
>130 END (RET)
>RUN (RET)
 100  101  102  103  104  105  106  107  108  109  110
 111  112  113  114  115  116  117  118  119  120  121
 122  123  124  125
>100 FOR X = 100000 TO 100025 (RET)
>110 PRINT X; (RET)
>120 NEXT X (RET)
>130 END (RET)
>RUN (RET)
 100000   100001   100002   100003   100004   100005   100006   100007
 100008   100009   100010   100011   100012   100013   100014   100015
 100016   100017   100018   100019   100020   100021   100022   100023
 100024   100025
>100 FOR X = 1000000 TO 1000013 (RET)
>110 PRINT X; (RET)
>120 NEXT X (RET)
>130 END (RET)
>RUN (RET)
 1.E+06    1.000001E+06    1.000002E+06    1.000003E+06    1.000004E+06
 1.000005E+06    1.000006E+06    1.000007E+06    1.000008E+06    1.000009E+06
 1.00001E+06    1.000011E+06    1.000012E+06    1.000013E+06
>
```

Figure 1. Example of BASIC Packed Format

3. FUNCTIONS AND SUBROUTINES

FUNCTIONS

The standard functions that BASIC provides are listed under "Mathematical Functions" in Chapter 1. The manner in which they are used is very simple. To compute $y=\sqrt{1+x^2}$ the programmer would write

```
Y = SQR (1 + X ↑ 2) ↵
```

The expression enclosed in parentheses is called the argument. The other standard functions are used in this same way; that is, the function name is followed by the argument enclosed in parentheses, as shown by the format

function-name (expression)

Examples:

```
LOG(Y)
SIN(X ↑ 2)
ABS(X + Y + Z)
```

Two additional functions that are in the BASIC repertory but which have not been described, are INT and RND

INT

The INT (integer) function is used to determine the integer part of a number that might not be a whole number. Thus INT (7.8) is equal to 7. As with the other functions, the argument of INT may be any expression. INT always operates by truncating the fractional part, whether the number is positive or negative.

One use of INT is to round numbers to the nearest integer. If the value of X, for example, is positive, it may be rounded by using the statement INT(X+.5). If the value of X is negative, however, the statement INT(X-.5) must be used, the reason being that a number like -7.8 rounded is -8 not -7. INT can be used to round to any number of decimal places. For positive values of X, for example, the statement INT(100*X+.5)/100 will round X to two decimal places.

RND

The RND function is a pseudo-random number generator. When called, it will produce a floating-point number with a value between zero and one. For example, the command

```
PRINT    RND. ↵
```

would cause BASIC to print

```
.502793
```

When the function is called repeatedly, it will produce a sequence of pseudo-random numbers. For example, the command

```
PRINT    RND., RND., RND. ↵
```

would cause BASIC to print

```
.502793        .2311643        .3898417
```

The same sequence of psuedo-random numbers will occur in every program that uses the RND function, which is useful for debugging.

DEF

The DEF command permits the user to define a function so he will not have to repeat a formula each time he uses it in his program. The name of a defined function must be three letters, the first two of which are FN. Thus the user may define up to 26 functions with DEF commands, that have the format

step DEF FN letter-1(letter-2) = expression ↵

where

letter-1 is the third letter of the user-defined function name.

letter-2 denotes an unsubscripted variable that is initially set to the value of the argument used in the call for the user-defined function. letter-2 is set equal to the value of the expression and is also returned as the result of the call for the user-defined function.

expression may be any expression that can fit into one line. It may not include another user-defined function, but may include standard functions (like SIN and SQR) and may involve other variables besides the one denoting the argument of the function. However, the variables used in the expression must not be subscripted.

Examples:

```
25 DEF FNF(Z) = (Z*3.14159265/180) ↵
40 DEF FNL(X) = LOG (X) / LOG (10) ↵
```

Thus, step 25 defines FNF as the function "sine of Z degrees" and step 40 defines FNL as the function "log-to-the-base-ten of X".

The DEF command may occur anywhere in the program. In a program containing FNF as defined above, the variable Z takes on a new value each time the function FNF is called. The safest practice is to avoid using elsewhere in a program the same variable used in a DEF command to define a function.

As another example:

```
60 DEF FNX(X) = SQR (X*X + Y*Y) ↵
```

may be used to set up a function that computes the square root of the sum of the squares of X and Y. To use FNX, one might write the following:

```
>10 Y = 30 ↵
>20 S1 = FNX(40) ↵
```

In this case, S1 is set to the value 50 as the result of step 20.

It should be noted that one does not need DEF unless the defined function must appear at two or more locations in the program. Thus,

```
>10 DEF FNF(Z) = SIN(Z*P) (RET)
>20 P = 3.14159265/180 (RET)
>30 FOR X = 0 TO 90 (RET)
>40 PRINT X, FNF(X) (RET)
>50 NEXT X (RET)
>60 END (RET)
```

might be more efficiently written as

```
>20 P = 3.14159265/180 (RET)
>30 FOR X = 0 TO 90 (RET)
>40 PRINT X, SIN(X*P) (RET)
>50 NEXT X (RET)
>60 END (RET)
```

to compute a table of values of the sine function in degrees.

SUBPROGRAMS

The use of DEF is limited to those cases where the value of the function can be computed within a single BASIC statement. Often much more complicated functions, or perhaps even sections of a program that are not functions, must be calculated at several places within the program. For this, the GOSUB command may be useful, which has the format

step-1 GOSUB step-2

Example:

```
200 GOSUB 400 (RET)
```

The effect of the GOSUB command is exactly the same as a GOTO command except that BASIC notes where the GOSUB command is in the program. As soon as a RETURN command is encountered, the computer automatically goes back to the command immediately following the most recently executed GOSUB command. As a skeleton example:

```
>100 X = 3 (RET)
>110 GOSUB 400 (RET)
>120 PRINT U, V, W (RET)
>200 X = 5 (RET)
>210 GOSUB 400 (RET)
>220 Z = U + 2*V + 3*W (RET)
>230 GO TO 440 (RET)
>400 U = X*X (RET)
>410 V = X*X*X (RET)
>420 W = X*X*X*X+X*X*X+X*X+X (RET)
>430 RETURN (RET)
>440 END (RET)
```

When step 400 is entered by the GOSUB 400 in step 110, the computations in steps 400, 410, and 420 are performed, after which the computer goes back to step 120. When the subprogram is entered from step 210, the computer goes back to step 220 after step 430 is executed the second time.

4. PROGRAM PREPARATION AND EXECUTION

PROGRAM INPUT FROM THE TELETYPE

There are several methods for preparing a BASIC program for execution. The first method is to type the program directly into BASIC. This is usually the procedure used for small and medium sized programs. For example:

```
-BASIC (RET)
>100 PRINT "THIS IS A SORTING PROGRAM" (RET)
>110 INPUT N (RET)
  :
```

PROGRAM INPUT FROM PAPER TAPE

It is often convenient as well as economical to type longer programs on paper tape while off line. To do this, the teletype is placed in (1) LOCAL MODE, (2) HALF DUPLEX MODE, and (3) PAPER TAPE PUNCH ON. The statements are typed just as though the user were connected to the computer, with one exception. Following each line, the user must type a carriage return and a line feed. When connected to the computer, the user types only a carriage return and the computer performs the line feed.

If the user has punched his program on a paper tape, he can enter the text into BASIC with the following procedure:

```
-BASIC (RET)
>LOAD TELETYPE (RET)
```

After this, BASIC is waiting for the program; when the user turns on the paper tape reader, the program reads in.

BASIC is unable to distinguish typed characters from those that are read from the paper tape reader. Therefore, an alternate way of entering a paper tape is to turn on the paper tape reader without using the LOAD command:

```
-BASIC (RET)
(turn on reader)
>100 PRINT "THIS IS A SORTING PROGRAM"
>110 INPUT N
  :
```

PROGRAM ON FILE

If the program to be run has been previously prepared and is now located in a file on the disc, the user may type the command LOAD to bring his program into memory. The format of the LOAD command is

```
LOAD/file-name/(RET)
```

Examples:

```
LOAD/PAYROLL/(RET)
LOAD/SORT/(RET)
```

MISCELLANEOUS BASIC COMMANDS

REM

An important part of any computer program is the description of what it does, and what data should be supplied. One of the ways a program can be "documented" is by supplying remarks along with the program itself. BASIC provides this capability with the REM (remark) command. For example, consider a program that calculates mean and standard deviation according to the formula

$$\text{Mean} = \frac{\Sigma x_i}{N} \qquad SD = \sqrt{\frac{\Sigma(x_i)^2 - \frac{(\Sigma x_i)^2}{N}}{N-1}}$$

```
>100 REM MEAN AND STANDARD DEVIATION (RET)
>110 S = 0 (RET)
>120 Q = 0 (RET)
>125 INPUT N (RET)
>130 FOR I = 1 TO N (RET)
>140 INPUT A(I) (RET)
>150 X = X + A(I) (RET)
>160 Q = Q + A(I) * A (I) (RET)
>170 NEXT I (RET)
>180 PRINT "MEAN=" S/N (RET)
>190 D = SQR(((Q-(S * S) / N)) / (N-1)) (RET)
>200 PRINT "SD=" D (RET)
>210 END (RET)
```

END

An END command indicates the termination point of a program. When the END command is encountered, it causes BASIC to stop executing the program and to await further commands.

RUN

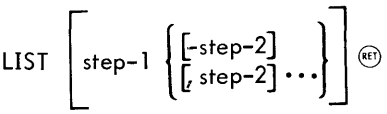
The user types RUN to begin execution. The program always begins with the smallest step number and executes according to ascending step numbers. To begin execution in the middle of the program, the user can use the GO TO command as a direct command.

STOP

The STOP command is used to stop program execution. When the program execution halts at the STOP command, the user may examine relevant variables. He may then, if he so desires, issue a command GO TO s, where s is the step number following the STOP command. The GO TO command will then cause execution to continue at step number s.

LIST

To list all or some of the steps of a program, the user types a LIST command, which has the format

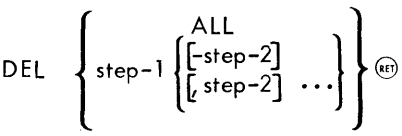


Examples:

LIST RET	(list entire program)
LIST 100 RET	(list step 100)
LIST 100-500 RET	(list steps 100 through 500)
LIST 100, 200, 300 RET	List steps 100, 200, and 300)

DEL

To delete one or more steps of a program, the user types a DEL command, which has the format



Examples:

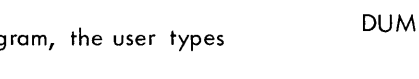
DEL ALL RET	(delete entire program)
DEL 100 RET	(delete step 100)
DEL 100-500 RET	(delete steps 100 through 500)
DEL 100, 200, 300 RET	(delete steps 100, 200, and 300)

To delete one step, the user may simply type the step number followed by a carriage return, e.g.,

> 125 RET

DUMP

To save a program on a permanent file, the user types a DUMP command, which has the format



Examples:

DUMP RET	(dump entire program)
DUMP 100-210 RET	(dump steps 100 through 210)
DUMP 100, 200, 221 RET	(dump steps 100, 200 and 221)

In response to the DUMP command, BASIC will print

ON:

on the line following the DUMP command. The user then types the name of the file on which he wishes to dump the desired portion of the program. The appropriate file name must appear between slashes, as follows:

/file-name/ RET

In response to the user's designation of the file, BASIC determines whether the file name currently exists in the user's file directory. If the name does exist, BASIC will print

OLD FILE

on the following line; however, if the name does not exist, BASIC will add the name to the user's file director and print

NEW FILE

on the following line:

Examples:

>DUMP RET
ON:/SAVE/RET
OLD FILE

>DUMP 100-120 RET
ON:/PGM/RET
NEW FILE

>DUMP 100, 200, 221 RET
ON:/P1/RET
OLD FILE

INDEX

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

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

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

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

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

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

600 East Bonita Avenue
Pomona, Calif. 91767
(714) 628-7371

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

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

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

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

SALES OFFICES

EASTERN

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

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

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

673 Panorama Trail West
Rochester, New York 14625
(716) 586-1500

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

SOUTHERN

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

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

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

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

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

MIDWEST

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

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

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

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

WESTERN

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

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

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

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

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

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

CANADA

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

INTERNATIONAL REPRESENTATIVES

FRANCE

Compagnie Internationale pour l'Informatique

EXECUTIVE OFFICES
101 Boulevard Murat
Paris 16eme

SALES OFFICES
17 Rue de la Reine
Boulogne 92

MANUFACTURING AND ENGINEERING
Rue Jean Jaures
Les Clayes Sous Bois 78

ISRAEL

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

JAPAN

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

90 11 11B