ABSTRACT

Data General's Single User BASIC is an implementation of the BASIC language as developed at Dartmouth College. Keyboard mode of operation has been implemented as an extension to the language.

Keyboard mode of operation gives the user immediate access to the central processor allowing him to perform two important programming functions with great efficiency. First, the user at the keyboard can request evaluation of simple desk calculator expressions and of complex formulas without writing a BASIC program.

User's Manual

PROGRAM

SINGLE USER BASIC

093-000042-02

TAPE

Absolute Binary: 091-000018

In addition, keyboard mode provides dynamic debugging facilities for BASIC programs. The user can interrupt a running program without destroying current values, examine and, if necessary, change current values, alter portions of the program if he wishes, and restart execution at the point of interruption or at another point in the program.

Ordering No. 093-000042-02

Printed in the United States of America
Rev. 02, June 1973

Original Release - September, 1970
First Revision - September, 1970
Second Revision - June, 1973

This revision of the Single-User BASIC Manual, 093-000042-02, supersedes revision 093-000042-01. The revision provides additional explanatory text and examples for use of BASIC and constitutes a major revision to the manual. The Single-User BASIC program is unchanged.

INTRODUCTION

Data General's Single-User BASIC allows programming in the standard BASIC language using a Nova or Supernova central processor with 4K or more of memory words and a teletypewriter. It includes use of all the elementary and advanced BASIC statements as defined in BASIC Programming by John G. Kemeny and Thomas E. Kurtz (c. 1967 by John Wiley & Sons, Inc.), but does not include matrix manipulation and string manipulation functions.

The BASIC language developed at Dartmouth College allows conversational program entry, editing, executing, and input/output operations. The Data General Corporation implementation also permits execution of certain statements in a "desk calculator" or "keyboard" mode of operation.

In keyboard mode of operation, the user has immediate access to the central processor; simple statements can be written and executed without writing a BASIC program. The statements follow the general format of certain BASIC program statements. Keyboard mode is useful for performing simple desk-calculator computations and for evaluating complex formulas. Immediate results are obtained, eliminating time usually spent on the writing and debugging of a BASIC program to obtain output of such computations.

An especially valuable feature of keyboard mode is that it is possible to interrupt an executing program without destroying the current values. The user can test current values of variables and change them if desired. He can examine the program or a portion of it and delete, insert, or change statements. He can add simple variables to the program. When he has concluded his changes, he can then resume execution at the point of interrupt or at some other point. Values altered by the user and the values unchanged during the interrupt will be the initial values at resumption of execution. In this way, keyboard mode of operation provides for interactive debugging of BASIC programs.

Data General's Single-user BASIC includes a comprehensive error detection system. As information is entered at the teletypewriter, it is examined for format and logical errors, and the appropriate error message is printed out if an error is detected. Additional errors discovered at run-time are printed out when detected.

TABLE OF CONTENTS

CHAPTER 3 - STATEMENTS (Continued)

CHAPTER 1

WRITING AND RUNNING A BASIC PROGRAM

PREPARING A BASIC PROGRAM

BASIC programs are made up of statements. Each statement is preceded by an integer that can be between 1 and 9999 inclusive. The number given a statement determines the order in which it is executed and listed. For example, two statements to be executed sequentially should be given sequential (but not necessarily consecutive) numbers.

Each statement is on a separate line. The programmer terminates each line at the teletypewriter with a carriage return (RETURN).

Typing errors on the teletype can be corrected by using special control keys:

1. Pressing CTRL and A at the same time erases the last character typed. A back arrow (←) is printed, representing the erasure.

2. Pressing CTRL and X at the same time deletes the entire line. A back slash (\ ) is printed, representing the line deletion. The programmer can continue typing a new statement without pressing carriage return.

An example of a BASIC program is given below. The example will be described in detail later in this chapter.

```
100 READ A,B,D,E
110 LET G = A*E-B*D
120 IF G = 0 THEN 180
130 READ C,F
140 LET X = (C*E-B*F)/G
150 LET Y = (A*F-C*D)/G
160 PRINT X,Y
170 GOTO 130
180 PRINT "NO UNIQUE SOLUTION"
190 DATA 1,2,4
200 DATA 2, -7, 5
210 DATA 1,3,4,-7
```

In the program, single letters represent program variables. A variable can be a single letter (e.g., Z) or a single letter followed by a single digit (e.g., Z4).

A BASIC program terminates when there are no more program statements or when an END or STOP statement is executed.

Most programs can be reduced to three steps:

1. Provide data.
2. Perform calculations.
3. Print answers.

Providing Data

One method to provide data is simply to write equations that contain the necessary values. The BASIC statement used for equations is the LET statement; for example:

```
20 LET X = 3.141  * 10.2        ; * means multiply
```

The statement will cause 3.141 and 10.2 to be multiplied and the resulting value will be stored in a variable named X.

However, writing values into equations is not very efficient. Programs are generally used for repetitive computations with a large number of different values. Instead of writing values into the equation, BASIC uses variables that can be assigned different values:

```
20 LET X = 3.141  * Y
```

To provide values, BASIC uses two statements, READ and DATA. The READ statement indicates the variables that are to have values and the DATA statement gives the values:

```
10 READ Y
20 LET X= 3.141  * Y
30 DATA 10.2, 7.3, -56.11, -.003, 3.4
```

There are now five possible values that Y will assume, which are listed in the DATA

Providing Data (Continued)

statement. Upon execution, the order of the values in the DATA statement is the order in which values are assigned to a variable or to several variables given in READ statements in the program.

Repetitive Computations

In general, statements in BASIC programs execute in the sequential order indicated by their statement numbers. However, if a program is completely sequential, it is not possible to perform repetitive calculations on a number of input values. For example, in the program given under the "Providing Data" section:

```
10 READ Y
20 LET X = 3.141  * Y
30 DATA 10.2, 7.3, -56.11, -.003, 3.4
```

Five data values are given for Y, but only the first one, 10.2, will be used because the program is completely sequential. It is necessary to insert a statement that will allow the READ and LET statements to be executed more than once:

```
25 GOTO 10
```

The GOTO statement causes a transfer back to statement number 10. The program "reads in" the second value for Y, 7.3, and executes the LET statement again. The program will continue to loop in this way until it runs out of data values for Y.

Note that the GOTO statement was given statement number 25. The reason why most BASIC programs are not numbered consecutively is to allow the programmer to insert any statements he may need without rewriting the entire program.

The GOTO statement is a means of transferring control to a part of a program in a non-sequential manner. There are several ways to do this in BASIC. Another useful statement in transferring control is the IF statement.

```
10 READ Y
15 IF Y <= 0 THEN 10              ; 10 means statement number 10
20 LET X = 3.141  * Y
25 GOTO 10
30 DATA 10.2, 7.3, -56.11, -.003, 3.4
```

Transfer in an IF statement depends upon whether the expression following the word IF is true or false. The expression is relational and uses the following symbols.

Relational Symbol	Meaning
=	Equal to
<=	Less than or equal to
<	Less than
>=	Greater than or equal to
>	Greater than
<>	Not equal to

The IF statement in the example would prevent the LET statement from being executed when the value of Y is zero or negative. The LET statement would only be executed for positive values of Y; otherwise, control would transfer back to the READ statement to read in another value.

Performing Calculations

The data provided as input must be computed into answers. A simple arithmetic statement of the calculations to be performed must be written in such a way that the BASIC system can recognize the operations required. The statement used is the LET statement.

The LET statement is used to assign the result of a calculation to some variable. The calculation to be evaluated, called an expression, appears on the righthand side of the equals sign in the LET statement. The variable to which the expression is assigned appears on the lefthand side of the equals sign. In the previous example, the expression provides for multiplying 3.1416 by some value assigned to the variable Y and then assigning the resultant value to the variable X.

The BASIC arithmetic symbols used in performing calculations are:

Symbol	Operator	Example	Meaning
+	Addition	A+B	Add B to A.
	Plus	+A	Positive A.
-	Subtraction	A-B	Subtract B from A.
		-A	Negative A.
*	Multiplication	A*B	Multiply A by B.

Symbol	Operator	Example	Meaning
/	Division	A/B	Divide A by B.
↑	Exponentiation	A↑B	Raise A to the power B (A^B) *

An expression is made up of elements described in Chapter 2 -- simple variables numbers, arrays, array elements, and functions which are linked together by the arithmetic symbols and by parentheses.

Parentheses may be used in arithmetic expressions to enclose subexpressions that are to be treated as entities. A subexpression in parentheses is evaluated first. Within each subexpression, arithmetic operations are performed in the sequence -- exponentation first, multiplication and division next, addition and subtraction last.

When two operations are of equal precedence, such as addition and subtraction, and there are no parentheses, evaluation proceeds from left to right in an expression.

In addition to arithmetic operations involving the arithmetic symbols, BASIC has a number of standard mathematical functions. These are described in Chapter 2; a few examples are:

SIN(X) Sine of X, where X must be in radians.

EXP(X) Natural exponential of X, e^X.

ABS(X) Absolute value of X.

An example of an expression to be evaluated and assigned to a variable is:

```
100 LET S = S - (17+SIN(Z))/3
```

* Single User BASIC computes A↑B by means of the identity A↑B = EXP(B*LOG(A)). This may result in some slight arithmetic errors in the sixth decimal position of the result. In addition, an arithmetic error (ERROR 16) will result if A is negative. To raise variables and constants to integer exponents, multiplication should be used rather than exponentation if this slight arithmetic error is objectionable.

In the example,

1. SIN(Z) is calculated. (Functions evaluated first.)

2. Result of step 1. is added to 17. (Parenthesized expression)

3. Result of step 2. is divided by 3. (Division has higher precedence than subtraction)

4. Result of step 3. is subtracted from the value of the variable S.

5. Result of step 4. is stored (=) into variable S as its new value.

Printing Output

The program is still not complete. It has data and performs calculations but the user has no way of knowing the results of those calculations. To complete the program, there must be a printout of results.

The PRINT statement is used to print out results of calculations. For example, if the program were written:

```
10 READ Y
20 LET X = 3.141  * Y
22 PRINT X
25 GOTO 10
30 DATA 10.2, 7.3, -56.11, -.003, 3.4
```

The PRINT statement is made part of the loop, so that a value for X is printed out each time the LET statement is executed. Each value of X will be printed out on a new line. The fact that the item X in the PRINT statement terminates with a carriage return means 'print the next value on a new line.' The output would look as follows:

```
 32.0382
 22.9293
   .
   .
   .
10.6794
```

Printing Output (Continued)

It is also possible to print out verbatim text using the PRINT statement. The user might want an explanation of each value printed. For example, the program could be written:

```
10 READ Y
20 LET X = 3.141  * Y
22 PRINT "FOR Y = ";Y;" X = ";X
25 GOTO 10
30 DATA 10.2, 7.3, -56.11, -.003, 3.4
```

The verbatim text is enclosed in quotation marks. It will be printed out exactly as shown with the same number of blank spaces. The semicolons between the items in the PRINT statement list mean 'print on the same line without spacing'. The output would now be printed as:

```
FOR Y = 10.2  X = 32.0382
FOR Y = 7.3  X = 22.9293
   .
   .
   .
FOR Y = 34.6  X = 10.6794
```

EXAMPLE OF A BASIC PROGRAM

A BASIC program for solving simultaneous linear equations would be:

```
100 READ A,B,D,E                    ;obtain values for variables
110 LET G = A*E-B*D                 ; evaluate denominator
120 IF G = 0 THEN 180               ; if G is 0, there is no unique solution
130 READ C,F                        ;obtain remaining variable values
140 LET X = (C*E-B*F)/G             ; solve for X
150 LET Y = (A*F-C*D)/G             ; solve for Y
160 PRINT X,Y                       ; print solutions for X and Y
170 GOTO 130                        ; loop to new values for C and F
180 PRINT "NO UNIQUE SOLUTION"      ; message printed if G = 0
190 DATA 1,2,4                      ; data for A, B, and D
200 DATA 2, -7, 5                   ; data for E and first values for C and F
210 DATA 1, 3, 4, -7                ; other values for C and F
```

EXAMPLE OF A BASIC PROGRAM (Continued)

The program solves the following equations:

x+2y = -7	x+2y = 1	x+2y = 4
4x+2y = 5	4x+2y = 3	4x+2y = -7

The program prints the paired X-Y values for each set of equations and issues an error message after printing the third pair.

Note that READ and DATA must both be included to provide input data for the BASIC program. The division of values among the data statements is arbitrary as long as the values are in the correct order. The programmer could have written the DATA statement as:

```
190 DATA 1,2,4,2
200 DATA -7, 5
210 DATA 1,3,4, -7
```

or as:

```
190 DATA 1,2,4,2,-7,5,1,3,4, -7
```

The blank spaces used in the BASIC program are only for readability. They could have been omitted. For example, the following statements are equivalent:

```
120 LET G = A * E - B * D

120 LETG=A*E-B*D
```

Within quotation marks, however, blanks in text are significant. If the programmer had written statement 180 as:

```
180 PRINT "NOUNIQUESOLUTION"
```

the resultant message (if G had been 0) would have been:

```
NOUNIQUESOLUTION
```

EDITING A PROGRAM

Besides the erase-character and delete-line features that make it possible to correct typing errors on input, the BASIC system has a number of keyboard commands that can be used to edit a BASIC source program not only during initial preparation but after input is complete or at an interrupt in execution.

All program control commands are described in Chapter 4. However, some of the simple commands used for editing are described here; the) symbol represents carriage return.

LIST) - causes a printout of the statements in the program in numeric order.

nLIST) - causes a printout of the statements in the program starting at the statement numbered n.

NEW) - clears the program, deleting all statements.

n new-statement) - replaces the statement on line n with new-statement, or if n is not in the program, inserts the new line at the proper point.

n) - deletes the statement with the line number n.

RUNNING A BASIC PROGRAM

When the programmer has written and edited his program, he can cause execution by giving the command:

```
RUN )
```

The program will be run from the lowest numbered statement. If no fatal program errors occur (Appendix A), the BASIC system will print out any output from the program and give the prompt "*READY" when execution is complete.

The programmer can cause a part of a program to be executed. If he types the commands:

```
GOTO n )
RUN )
```

the program will transfer control to statement n where it will begin execution.

INTERRUPTING A RUNNING PROGRAM

If a program is executing without error indications but the output appears to be incorrect, the programmer can stop execution by pressing the ESC key on the teletypewriter. Execution will cease and the programmer can now change his program using keyboard editing commands.

The ESC key is the "change mode" key. In interrupting execution, the programmer is changing from program mode to keyboard mode. The ESC key can be used for other types of mode change; for example, it can be used to stop an excessively long listing after a LIST command has been given.

CHAPTER 2

ARITHMETIC OPERATIONS

ARITHMETIC OPERATIONS

Numbers

Numbers handled by Single-user BASIC have the following limits:

Input	-	7 significant digits, e.g. 45678.91
Output	-	6 significant digits, e.g., -1.33333 or 1.61333
Internal	-	23 significant bits
Range	-	Positive powers: $\pm (1-2^{-23})*(2^{127}) \cong \pm 1.70141 \times 10^{38}$
		Negative powers: $\pm (1/2)*(2^{-127}) \cong \pm 2.0 \times 10^{-39}$

Any real or integer number that consists of six or less digits is printed out without using exponential form. A real or integer number that requires more than six digits will be printed out in 6-digit format, followed by the letter E, followed by an exponent.

Number Represented	Output Format
2,000,000	2.00000E+6
20,000,000,000	2.00000E+10
108.999	108.999
.0000256789	2.56789E-5
25	25
.16	.16
1/16	.0625

Arithmetic Variables

The names of arithmetic variables are either a single letter or a single letter followed by a single digit: A A3 X Z5

Arithmetic Expressions

Arithmetic expressions can be composed of simple variables, arrays, array elements, and functions linked together by parentheses and by the arithmetic operators. The arithmetic operators are:

SYMBOL	EXAMPLE	MEANING
+	X+Y	Addition (add X to Y)
+	+X	Plus (positive X)
-	X-Y	Subtraction (subtract Y from X)
-	-X	Minus (negative X)
*	X*Y	Multiplication (multiply X by Y)
/	X/Y	Division (divide X by Y)
↑	X↑Y	Raise to the power (find X^Y) *

The order in which operations are evaluated affects the result. In BASIC, unary minus or plus is evaluated first, then exponentiation, then multiplication and division, and last addition and subtraction. When two operations are of equal precedence (* and /), evaluation proceeds from left to right. For example:

```
Z - A + B * C ↑ D
```

1. C↑D is evaluated first.
2. B is multiplied by the value from 1.
3. A is subtracted from Z.
4. The value from 2. is added to the value from 3.

The programmer can change the order of evaluation by enclosing subexpressions in

---

* In Single User BASIC an error results from an attempt to raise a negative number to a power. If the exponent is an integer less than 2^{15} in absolute value, exponentation is performed by multiplications; otherwise, the LOG and EXP functions are used, i.e., EXP(Y*LOG(X)) = X↑Y.

parentheses. A parenthesized expression is evaluated first. Parentheses can be nested, and the inmost parenthesized operation is always evaluated first. For example:

```
Z - ( ( A + B ) * C ) ↑ D
```

1. A + B is evaluated first.
2. The value from 1. is multiplied by C.
3. The value from 2. is raised to the power of D.
4. The value from 3. is subtracted from Z.

Some examples of expressions are:

```
L1 + 1
INT(C/D)/10
(Z(J. J)*Z(I, J))/A * (ABS(I))
J-5
SQR(ABS(X))
```

Arrays

An array represents an ordered set of values. Each member of the set is an array element. Names of arrays are written as a single letter (A-Z). The letter must be unique; it cannot also be used as the name of a variable in the program or an error message will result. An attempt to dimension a variable name, such as Z3, will also cause an error.

Declaring an Array

Most arrays are declared in a DIM statement, which gives the name of the array and its dimensions. An array can have either one or two dimensions. The lower bound of a dimension is always 0; the upper bound is given in the DIM statement. Dimensioning information is enclosed in either parentheses or square brackets, following the name of the array:

```
5 DIM A(15), B[2, 3]        ;A is a one-dimensional array of 16 elements (0-15).
                            ;B is a two-dimensional array of 12 elements.
```

Declaring an Array (Continued)

If the programmer uses an array but does not declare it in a DIM statement, BASIC sets aside 11 elements (0-10) for each dimension. An undeclared one-dimensional array cannot have more than 11 elements. If the programmer does not need 11 or 121 elements for a given array and wishes to conserve space, he should declare the array with the required number of elements. Each dimension of an array must be less than or equal to 256 elements. An array must be less than or equal to 1024 elements. An error message will result if the limit of array size or of dimension size is exceeded.

Array Elements

Each of the elements of an array is identified by the name of the array followed by a parenthesized subscript. (The subscript could, alternatively, be enclosed in square brackets). The elements of array B[9] would be:

```
B(0), B(1), B(2), B(3), B(4), B(5), B(6), B(7), B(8), B(9)
```

For a two-dimensional array, the first number gives the number of the row and the second gives the number of the column for each element. The elements of array C(2,3) would be:

```
C(0,0)   C(0,1)   C(0,2)   C(0,3)
C(1,0)   C(1,1)   C(1,2)   C(1,3)
C(2,0)   C(2,1)   C(2,2)   C(2,3)
```

Values are stored into elements of a two-dimensional array be filling each row before beginning the next. The order in which elements of array C would be filled is:

```
C(0,0), C(0,1), C(0,2), C(0,3), C(1,0),...,C(2,2),C(2,3)
```

An array element can be referenced with integer or expression subscripts. Any expression or variable that can be evaluated by the BASIC system producing a value greater than or equal to 0 can be used as a subscript. If the variable or expression does not evaluate to an integer, the BASIC system will convert it to

fixed form using the INT function, described in the section on functions starting on page 2-6. For example, some elements of array E(24, 5) might be:

```
E(I-3, J*K)
E(0, 5)
E(ABS(R), 5)          ;ABS is a function described later in this chapter.
```

If a subscript evaluates to an integer larger than the limit of the dimension for the array, an error message will be printed.

Redimensioning Arrays

It is possible to redimension a previously defined array during execution of a program. Redimensioning does not affect the amount of storage previously defined for the array nor the current contents of the array.

Redimensioning is used primarily to change the subscripting of two-dimensional arrays. Suppose the user originally defines a 3x4 array A.

```
100 DIM A(2, 3)
```

Statement defining A.

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Row/column assignment of values to elements of array A. A(0, 0) contains 1, A(0, 1) contains 2, ..., A(2, 2) contains 11, and A(2, 3) contains 12.

Later the user might redimension A using the keyboard command DIM. (See Chapter 4).

DIM A(3,2)

Command transposing the dimensions of A.

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

Row/column assignments of values to elements of A. The values remain the same but the subscripts required to retrieve those values have changed:

Value	New Subscript	Old Subscript
4	A(1,0)	A(0,3)
6	A(1,2)	A(1,1)
8	A(2,1)	A(1,3)
11	A(3,1)	A(2,2)

An array can only be redimensioned so that it has the same or fewer elements. For example, redimensioning a 3 x 5 array as a 4 x 4 array will cause an error.

Subscript references outside the defined range of subscripts will cause errors. For example, once array A above is redefined as A(3,2), use of 3 as a column subscript (e.g., A(2,3)) will cause an error.

Redimensioning an array to have fewer elements (e.g., redimensioning B(3,5) as B(4,3) or redimensioning C(20) as C(15)) merely makes referencing the unused locations impossible. It does not free the locations for other storage.

FUNCTIONS

Some of the examples shown before contained functions. Certain standard mathematical functions are supplied as part of the BASIC system. They are:

SIN(X)	Find the sine of X, where X is in radians.
COS(X)	Find the cosine of X, where X is in radians.
TAN(X)	Find the tangent of X, where X is in radians.
ATN(X)	Find the arctangent of X, where X is in radians.
LOG(X)	Find the natural logarithm of X.
EXP(X)	Find e^X.
SQR(X)	Find the square root of X.
ABS(X)	Find the absolute value of X.

The arguments of SIN, COS, TAN, ATN and ABS are confined to the range of acceptable real numbers ($2 \times 10^{-39} \leq |x| \leq 1.7 \times 10^{38}$).

The LOG and SQR functions require positive arguments. A negative or zero argument in the LOG function will cause the system to respond with the largest possible negative number (-1.70141E+38) plus an error message. A negative argument in the SQR function will cause the system to respond with the square root of the same positive number plus an error message.

The argument of the EXP function is confined to those values which will generate the largest and smallest acceptable real numbers, e.g., $e^{88} = 1.7 \times 10^{38}$, so that arguments greater than 88 will cause the system to return 1.70141E+38 as the answer with an error message.

In addition to the standard mathematical functions, the following functions are supplied as part of the BASIC system.

INT(X)	Find the greatest integer not larger than X.
RND(X)	Find a random number between 0 and 1.
SGN(X)	Find the algebraic sign of X.

The INT function yields the largest integer less than or equal to its argument. If the argument has an absolute value greater than 2^{31} $(2,147,483,648_{10})$, it will be reduced to the value $\pm$ 2.14748E+9.

```
INT(7.25)     =     7
INT(12)       =     12
INT(-.1)      =     -1
INT(X+2)      =     16      ;if X evaluates to 14.9
INT(1.5)      =     1
```

INT may be used to round a number to the nearest integer. To round the value, add 0.5 to the argument:

```
INT(X+0.5)
```

The RND function yields a random number having a value between 0 and 1. The function requires an argument, although the argument does not affect the resulting random number. The argument can be any constant or previously defined variable.

```
RND(1)      might produce     .654318

RND(0)      might produce     .004561
```

The SGN function generates its result as +1 if the argument is positive, 0 if the argument is 0, and -1 if the argument is negative.

```
SGN(.452)     =          1

SGN(0.00)     =          0

SGN(-24.8)    =          -1
```

CHAPTER 3

STATEMENTS

ELEMENTARY AND ADVANCED BASIC STATEMENTS

As shown in the first chapter, only a few BASIC statements are needed in order to write a simple BASIC program. Those statements needed for elementary BASIC programming and those available for advanced BASIC programming are usually divided into the groups shown below.

Elementary BASIC Statements	Advanced BASIC Statements
LET	All elementary statements.
READ and DATA	INPUT
GOTO	GOSUB and RETURN
IF-THEN	DEF
FOR and NEXT	REM
DIM	PRINT (advanced formatting)
END	RESTORE
PRINT (simple formatting)	STOP

In this chapter all statements, whether elementary or advanced, are given in alphabetical order.

DEF

Format:

DEF FNa (d) = expression

where: a is a single letter, A-Z.

d is a dummy variable that may appear in expression.

Purpose: To permit a user to define a function that can be referenced several times during a program. The function returns a value to the point of reference.

When a function is referenced, the constant, variable, or expressions appearing in the reference argument replaces the dummy argument d in the expression.

In the function definition, expression can be any legal expression including one containing other user-defined functions. However, if functions are nested too deeply, an error will result.

Function definition is limited to those formulas that can be expressed on a single line of text. For longer formulas, subroutines should be used.

Examples:

```
100 DEF FNE (X) = EXP (X*2)          ;definition of the function

200 LET Y = Y*FNE(.1)                ;function reference; argument = .1

300 IF FNE(A+3) < Y THEN 150         ;function reference, argument = A+3
```

```
30 LET P = 3.14159
40 DEF FNR(X) = X*P/180
50 DEF FNS(X) = SIN(FNR(X))          ;Function FNR is nested in FNS.
70 FOR X = 0 TO 45 STEP 5
80 PRINT X, FNS(X)                   ;FNS is referenced with X having values
90 NEXT X                            ;0,5,10,...,45
```

DIM

Format:

DIM $array_1(dims), \ldots array_n(dims)$

Purpose: To give the dimensions of one and two dimensional arrays. The information in this non-executable statement is used to allocate storage.

Arrays are dimensioned as follows:

1. The lower bound is always 0 and does not appear in the DIM statement.

2. The upper bound is given in parentheses or square brackets following the array name.

3. If there are two upper bounds, the bounds are separated by a comma.

Arrays may appear in any order in a DIM statement.

Example:

```
2 DIM A(5,6), C(20), X(17), Y(14,10)
```

A is a 6x7-element two dimensional array.
C is a 21-element one dimensional array.
X is a 18-element one dimensional array.
Y is a 15x11-element two dimensional array.

END

Format:

```
END
```

Purpose: Many BASIC systems require an END statement as the last program statement or as the terminating statement of a main program that calls one or more subroutines. In data General's BASIC, main programs and subroutines terminate at the last logically executed statement in the program (if an END statement or STOP statement is not encountered). However, the implementation allows END statements for compatibility with BASIC programs written for other systems.

FOR and NEXT

FOR

Format:

```
FOR control variable = expression1 TO expression2

FOR control variable = expression1 TO expression2 STEP
                                                expression3
```

Purpose: To establish beginning, terminating and incremental values for control variable, a variable that determines the number of times statements contained in a loop are to be executed.

The loop consists of statements following the FOR statement up to a NEXT statement that contains the name of control variable. The variable in a FOR statement cannot be subscripted.

$expression_1$ is the first value of the variable.
$expression_2$ is the terminating value of the variable.
$expression_3$ is the increment added to the variable each time the loop is executed. If not given, the increment is +1.

When the NEXT statement containing the variable name is encountered. the loop is executed again. The looping ends and the statement after NEXT is executed when control variable exceeds the terminating value, $expression_2$.

FOR loops may be nested to a depth of four. The FOR statement and its terminating NEXT statement must be completely nested. For example:

Legal Nesting

```
FOR X = ...
  FOR Y = ...
    FOR Z = ...
    NEXT Z
  NEXT Y
NEXT X
```

Illegal Nesting

```
FOR X = ...
  FOR Y = ...
  NEXT X
NEXT Y
```

NEXT

Format:

```
NEXT control variable
```

Purpose: To terminate the loop beginning with a FOR statement. The control variable contained in the NEXT statement must precisely match the control variable contained in the last uncompleted FOR statement preceding NEXT.

When the FOR statement conditions have been fulfilled, execution continues at the statement following NEXT.

FOR AND NEXT EXAMPLES

```
5 FOR X = .1 TO .005 STEP -0.01
10 LET X = X*LOG(X)
20 NEXT X
```

```
10 FOR I = 1 TO 45
20 PRINT 235*I
30 NEXT I
```

```
100 FOR I = 1 TO 3
120 FOR J = 1 TO 20 STEP I
130 READ B(I, J)
140 NEXT J
150 NEXT I
```

J Loop

I Loop

```
90 FOR I = 1 TO 9
100 NEXT I
110 PRINT I
RUN )
9                          ;final value of I loop is the terminating value, 9.
120 FOR J = 1 TO 9 STEP 3
130 NEXT J
140 PRINT J
RUN )                      ;final value of the J loop is the last value before the
7                          ;terminating value is exceeded.
```

GOSUB and RETURN

GOSUB

Format:

```
GOSUB statement number
```

Purpose: To transfer control to statement number, the first statement in a subroutine.

A portion of a program is written as a subroutine when it is executed repetitively, possibly at several different places in the program, and when it is of greater complexity than a simple loop. GOSUB statements may be nested to a depth of four, i.e., one subroutine may call another subroutine or may call itself.

RETURN

Format:

```
RETURN
```

Purpose: To exit a subroutine, returning to the first statement after the GOSUB that had caused the subroutine to be entered.

EXAMPLES OF GOSUB AND RETURN

In the example following, RETURN causes return to statement number 120 when the subroutine is entered from statement 110; return is made to statement 140 when the subroutine is entered from statement 130, etc.

```
100 LET X = 5
110 GOSUB 500
120 LET X = 7
130 GOSUB 500
140 LET X =11
150 GOSUB 500
160 STOP
    :
500 LET Y = 3*X
510 LET Z = 1.2 *EXP(Y)
520 LET Y = SQR(Z+2)
550 PRINT X, Y
560 RETURN
```

GOTO

Format:

GOTO statement number

Purpose: To transfer control to a statement that is not in normal sequential order. If control is transferred to an executable statement, that statement and those following will be executed. If control is transferred to a non-executable statement (e.g., DATA), the first executable statement following the one to which transfer was made will be executed.

Examples:

```
200 READ X, Y, Z
220 LET A = SQR(X ↑2 + Y ↑2 - 2 * X * Y * FNC(Z))
230 PRINT X, Y, Z, A
240 GOTO 200                    ;control will continue to transfer back to
      :                         ;statement 200 until all values for X, Y,
      .                         ;and Z have been read.
```

```
190 DATA 19, -5, -2, 5, -6, 10, 10, 60, 20, 5, 50, 10
200 READ X, Y, Z
220 LET A = SQR (X ↑2 + Y ↑2 - 2 * X * Y * FNC(Z))
230 PRINT X, Y, Z, A
240 GOTO 190                    ;same as the previous example.
```

IF

Format:

IF relational-expression THEN statement-number

IF relational-expression GOTO statement-number

Purpose: To transfer control to the statement having statement-number if the relational-expression is true. If relational-expression is not true, control is passed sequentially to the next statement following the IF statement.

IF-THEN and IF-GOTO are equivalent forms.

Relational Expression: A relational expression consists of two arithmetic expressions and a relational operator and has the form:

$expression_1$ relational operator $expression_2$

The relational operators are:

Symbol	Meaning	Example
=	Equal to	A = B
<	Less than	A < B
<=	Less than or equal to	A <= B
>	Greater than	A > B
>=	Greater than or equal to	A >= B
<>	Not equal to	A <> B

A simple expression may be used in place of relational expression following IF. A simple expression is an arithmetic variable or an arithmetic literal. A simple expression is assumed to be true if it does not evaluate to 0. Any non-zero arithmetic expression that contains one or more characters will be evaluated to true.

Example:

```
100 IF X + Y = 0 THEN 100
150 IF .01 < = SQR(X) GOTO 410
```

Relational expressions, where values are compared to determine the truth value.

INPUT

Format:

INPUT variable list

where: variable list can contain arithmetic variables or array elements.

Purpose: To input values for variables at run time. When an INPUT statement in the program is executed, the BASIC system types ? at the teletypewriter, requesting data for the variables.

The programmer types the list of data values for input immediately following the symbol ? . Each datum is delimited from the next by either a comma or a carriage return. The last data value for the INPUT list must terminate with a carriage return.

Arithmetic variables and array elements may be interspersed in the variable list of the INPUT statement. The data list typed by the programmer must match the variable list in number of data items.

Pressing the carriage return during typing of the data list does not cause an actual carriage return; it merely signals BASIC that a value has been terminated. If the return is pressed and BASIC does not have enough values to fill the input list, the BASIC system types:

?

and sounds the Teletypewriter bell. The programmer can then add one or more values to the list.

If the data list contains an error detected by the system, the BASIC system will type:

/?

The programmer can then retype the corrected value. If the error was detected in a data value in a list where items are delimited by commas, the list must be retyped in its entirety. If the error was detected in a list where a previous value or values were delimited by carriage returns, only the value in error need be retyped.

INPUT (Continued)

During the typing of the data list, the programmer may use the line erase (CTRL X) or character erase (CTRL A) to correct errors in the list.

It is useful to precede the INPUT statement with a PRINT statement that will clarify which variables the values are requested for.

If an INPUT statement is incorporated into a continuous loop, the programmer can terminate program execution and the printing of ? by pressing the ESC key.

Examples:

```
20 PRINT "VALUES OF X, Y, Z"
30 INPUT X, Y, Z
                .
                .
                .
RUN )

VALUES OF X, Y, Z
? 2.5, -44.1, .5              ;values separated by commas

RUN )

VALUES OF X, Y, Z
? 2.5 ? -44.1 ? .5            ;values separated by carriage returns (invisible).
```

```
10 PRINT "VALUES OF X, Y, Z "
20 INPUT X
30 INPUT Y
40 INPUT Z

RUN )

VALUES OF X, Y, Z
? 2.5, -44.1, .5              ;error message will result since only one value is expected.
```

LET

Format:

```
LET variable = expression
```

Purpose: To evaluate expression and assign the value to variable.

The variable may be subscripted.

Examples:

```
10 LET A = 4.17 + G

40 LET X = X + Y

80 LET W7 = ((W-X) *234) * SQR(Z-A)/B

90 LET J(I, K) = COS(FNA(K+I))
```

PRINT or ;

Format:

```
PRINT expression list

; expression list
```

where: expression list is a list of numeric variables, subscripted variables, expressions, or string literals.

PRINT and ; are equivalent statement forms.

Purpose: To output current values for any expressions and variables appearing in the expression list of the PRINT statement, and to output verbatim text for any string literals in the expression list.

Format: The PRINT statement allows the user to either control output formatting or to accept default formatting.

Number Representation

Any real or integer number less than or equal to 6 digits is printed out without using exponential form. A minus sign is printed if the number is negative; a space is left before a positive number.

All other numbers are printed in the format:

[-]n.nnnnnE+e[e]

where: n is a digit.
E indicates exponentation.
e is a digit of the exponent
[ ] square brackets indicate optional parts of the number. If the number is positive, a space is left in the sign position.

Number	Printed Output
.00000002	2.00000E-8
-.0002	-.0002
200	200
-200.002	-200.002
2,000,000	2.00000E+6
-20,000,000,000	-2.00000E+10

Format:
(Continued)

Zone Spacing of Output

The teletypewriter line is divided into five zones, which are set at either of the following character positions:

```
0      15     30     45     60     (15-space zone)
0      14     28     42     56     (14-space zone)
```

Zoning is set at load time by replying to the system query:

DO YOU WISH COMMA'S TO BE 14 SPACES (TYPE Y) OR 15 SPACES (TYPE N) ?

A comma between items in the expression list of the PRINT statement indicates "space to the next zone". If the fifth print zone has been filled, the next value is printed in the first print zone of the next line.

```
10 LET X = 5                              (Assume a 14-space zone.
30 PRINT  X, (X*2)↑6, X*2,                Note terminating comma
60 PRINT X↑4, X-25, (X*2)↑8, X-100        on first PRINT statement
                                          controls the output of the
                                          first value of the next
                                          PRINT statement.)

0              14             28            42            56
↓              ↓              ↓             ↓             ↓
 5              1.00000E+6     10            625          -20
 1.00000E+8    -95
```

When an output value is longer than a single zone, for example, a long string literal, the teletype is spaced to the next free zone to print the next value.

```
10 LET X = 25
20 PRINT "THE SQUARE ROOT OF X IS:", SQR(X)

0              14             28
↓              ↓              ↓
THE SQUARE ROOT OF X IS:       5
```

Format:
(Continued)

Compact Spacing of Output

The user can obtain more compact output by use of the semicolon between list items. It inhibits spacing to a print zone, leaving only a single space between the values output for list items. Note that like the comma, a semicolon at the end of a PRINT statement will determine the position of the first value of the next PRINT statement.

```
10 LET X = 5
20 PRINT X; (X*2)↑6; X*2; (X*2)↑4;
30 PRINT X-25; (X*2)↑8; X-100

0   4              16   20      26   31             42
↓   ↓              ↓    ↓       ↓    ↓              ↓
 5  1.00000E+6    10   10000   -20   1.00000E+8    -95
```

Spacing to the Next Line

If there is no comma or semicolon terminating the last item of the list of a PRINT statement, the next value will be printed on the next line.

```
10 LET X = 5
20 PRINT X, (X*2)↑6
30 PRINT X * 2
40 PRINT X-25; (X*2)↑8
50 PRINT X-100

0      5             15
↓      ↓             ↓
 5                   1.00000E+6
 10
-20    1.00000E+8
-95
```

Format:
(Continued)

Tabulation

It is possible to tabulate to a particular character position to output a value using the TAB function:

```
TAB (expression)
```

where: expression evaluates to an integer representing the character position of the next list item following the TAB function. The TAB function only affects the next list item and only tabulates to the given position if:

1. The carriage is not set beyond the desired character position.

2. The function is not overridden by another formatting delimiter such as a comma.

```
10 DATA 5, -7, 9, -11
20 READ A,B,C,D
30 PRINT TAB(5);A;TAB(10);B;TAB(15);C;TAB(20);D

0     5     10    15    20
↓     ↓     ↓     ↓     ↓
      5     -7    9     -11
```

```
10 DIM B(4)
20 DATA 5, -7, 9, -11
30 FOR I = 0 TO 3
40 READ B(I)
50 ;TAB(5); B(I)
70 NEXT I

0     5    7    10   13
↓     ↓    ↓    ↓    ↓
      5    -7   9    -11     ;Note that the TAB function only
                               affects the first array value.
```

Format:
(Continued)

Tabulation (Continued)

The TAB function must be preceded by a semicolon. Otherwise, a syntax error will result:

```
70 PRINT B(I) TAB(5)
```

In the example above, the values of the array are each output on a separate line, and each is followed by a syntax error message.

If the expression in the TAB function evaluates to a number greater than the carriage length, the value will be printed on the line but the character position will vary mod(carriage length).

Following are a few additional examples of output printing:

```
10 FOR I = 1 TO 10
20 PRINT I                ;Carriage return delimiter.
30 NEXT I

1
2
3
.
.
.
10
```

```
10 FOR I = 1 TO 10
20 ; I,                   ;If zone = 15 spaces.
30 NEXT I

1          2          3          4          5
6          7          8          9          10
```

READ

Format:

```
READ variable list
```

where: variable list can contain arithmetic variables and array elements.

Purpose: To read values from the data block into variables listed in the READ statement.

The order in which variables appear in READ statements is the order in which values for the variables are retrieved from the data block.

Values appearing in all DATA statements in a program are stored, before a program is executed, into a single data block for use as values of variables in the READ statement.

Normally, READ statements are placed in the program at those points at which data is to be manipulated, while DATA statements may be placed anywhere.

A pointer is moved to each consecutive value in the data block as values are retrieved for variables in the READ statements. If the number of variables in the READ statement exceeds the number of values in the data block, an "out of data" error message is printed. The RESTORE statement can be used to reset the pointer to the beginning of the data block.

READ and DATA (Continued)

DATA

Format:

DATA number list

Purpose: To provide values to be read into variables appearing in the READ statements.

Only numbers may appear in DATA statements. Formulas and string constants will not be interpreted. Each number is separated from the next datum by a comma.

DATA is a non-executable statement. The values appearing in the DATA statement or statements are read into a single data block before the program is run.

EXAMPLES OF READ AND DATA

```
150 READ X, Y, Z
      .
      .
      .
200 READ A
      :
      :
250 FOR I = 0 TO 10
255 READ B(I)
260 NEXT I
      :
      :
400 DATA 4.2,7.5, 25.1, -1, .1, .01, .001, .0001
450 DATA .2, .02, .002, .0002, .015, .025, .3, .03, .003
```

The first three data values are read for X, Y, and Z respectively. The value -1 is read into A. The next eleven values, .1 through .3, are read into the eleven elements of array B.

EXAMPLES OF READ AND DATA (Continued)

```
        .
        .
        .
100 READ A,B,C
        .
        .
        .
300 GOTO 100
        .
        .
        .
500 DATA 1, 10, .333
510 DATA -1, 1, .555
520 DATA 0, -1, .1
```

Each series of data values, contained in the three DATA statements will, in turn, be read into variables A, B, and C.

REM

Format:

```
REM text comment
```

Purpose: To insert expanatory comments within a program. The text following REM is stored before the program is run and is reproduced exactly as it appears in the statement when a listing of the program is printed. Although the REM statement is non-executable, note that storage space is required for the text.

Example:

```
100 REM THIS IS A PROGRAM TO FIND COMPOUND INTEREST
```

RESTORE

Format:

```
RESTORE
```

Purpose: To permit reuse of the data block. RESTORE sets the data block pointer to the first value in the data block. The next READ statement following execution of a RESTORE statement will begin reading values from the start of the data block into variables.

Example:

```
20 FOR K = 0 TO 10
30 READ B(K)                          ;Data values 1,2,...11 read into ele-
40 NEXT K                             ;ments of array B
50 RESTORE
60 READ X,Y,Z                         ;Data values 1,2,3 read into X,Y,Z
70 RESTORE                            ;respectively.
      .
      .
      .
200 READ ----                         ;At next READ, values start at 1 again.
500 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13
```

STOP

Format:

```
STOP
```

Purpose: To halt execution of a program at some point. When STOP is encountered, the BASIC system will cease execution and type the message:

STOP AT xxxx

where: xxxx is the line number of the STOP statement.

The system will wait for a keyboard command.

Example:

```
80 FOR K = 0 TO M-1
90 LET X = B + K * P
100 IF X-M*INT(X/M) = A THEN 130
105 NEXT K
110 PRINT "ERROR"
120 STOP                                ;Stop program if error occurs.
130 LET P = P * M
            .
            .
            .
```

CHAPTER 4

KEYBOARD COMMANDS

Keyboard commands, typed by the user without statement numbers, are recognized by the BASIC system as commands to be immediately executed. Each command is terminated by a carriage return ( ).

The commands allow the user to edit, execute, perform dynamic debugging, and to perform simple desk calculator operations.

PROGRAM CONTROL

Program control is achieved through use of the special program control keys and the program control commands.

Program Control Keys

ESC

Pressing the ESC key essentially means "change the current mode of operation." The effect depends upon the current state of the system:

1. If a program is executing, execution ceases, and the message:

   STOP AT xxxx

   is printed, where xxxx is the statement number before which execution ceased.

2. If output is being listed, the listing halts at the end of the current line.

3. If input is being read from the paper tape reader, input processing ceases. (Paper tape input can only be restarted from the beginning of the tape.)

4. In keyboard mode, the user can press ESC instead of CTRL X to effect a line kill.

5. If the user wishes to issue a keyboard command and the system is operating in one of the modes indicated above, pressing the ESC key will change the mode to allow the system to accept a keyboard command.

Program Control Keys (Continued)

CTRL X — When the user is writing and editing BASIC programs at the keyboard and when he is responding to an INPUT request, pressing both the CTRL and X keys simultaneously results in deletion of the line he is currently typing. He may then retype the line without giving a carriage return.

The symbol \ is printed at the teletypewriter to indicate CTRL X. The following example shows line deletion and replacement.

90 PROMPT "OMTEREST\90 PRINT "INTEREST-";I

Typing Errors | Text Deleted | Replacement Statement

CTRL A — When the user is writing and editing BASIC programs at the keyboard and when he is responding to an INPUT request, pressing both the CTRL and A keys simultaneously results in the deletion of the last character in the current line. He may then retype the character.

The symbol ← is printed at the teletype to indicate CTRL A. The following example shows character deletion and replacement:

90 PRO ←INT "OM ← ← INTEREST @ 6% ←←5% IS: ";I

The statement within the program will appear as:

90 PRINT "INTEREST @5% IS: ";I

Program Control Commands

Program control commands can be roughly divided into those dealing with input/output program editing, and program execution.

Input/Output Commands

The keyboard commands used for I/O are:

NEW) - All currently loaded statements and variables are cleared. It is usual to give this command before beginning input processing of an entirely new program.

Program Control Commands (Continued)

Input/Output Commands (Continued)

If the NEW command is not given, incoming statements will replace currently loaded statements that have the same statement numbers, but already loaded statements with unique statement numbers will not be affected. This makes it possible to add statements and subroutines to a program without destroying that part of the program that is already loaded.

LIST) - Output a listing of the currently loaded program to the teletypewriter.

nLIST) - Output a listing of the currently loaded program to the teletypewriter beginning at statement numbered n and continuing through the end of the program.

When the BASIC system has completed any of the above commands, the system stops; no message is typed at the teletypewriter.

Editing Commands

The commands that may be used to edit programs (in addition to CTRL X and CTRL A control keys) are:

n) The statement number n is deleted. If there is no statement n, an error message is typed.

n statement) Replace the statement numbered n with the given statement.

Execution Command

When the user has written and corrected his BASIC program, he initiates execution by the command:

RUN) Clear all variables; un-dimension all arrays; do a RESTORE; then execute the program from its lowest numbered statement.

DESK CALCULATOR AND DEBUGGING COMMANDS

Several BASIC statements have been adapted for use as keyboard commands. They are DIM, GOTO, PRINT, LET, and RESTORE.

The commands can be used to perform calculations. Using the PRINT command, the user can simply request the value of a formula, given by literals. The command can also be used to perform calculations using current program values of a loaded program. As an example, the user might interrupt a running program that contains an array to which values have been assigned and use those values to obtain a series of random numbers by multiplying the array elements by the RND function.

Finally, the commands provide an easy-to-use method of dynamic program debugging. A running program can be interrupted at different program points; the current values of the variables can be checked at those points and changed as necessary. To resume execution of a program following a programmed STOP or keyboard interruption, the user must type "GOTO xxxx" where xxxx is the line number of the statement at which the program should be reentered. If the user types "RUN" all variables and arrays in the program will be re-initialized to zero, a RESTORE will be executed on the data block, and the program will be started again from the lowest numbered statement. By typing the GOTO command it is then possible to restart the interrupted program at any point without losing either the values of the variables at the point of interruption or those values that were inserted or changed during the interrupt.

PRINT Command

The keyboard PRINT command, typed either as ; or PRINT, can be followed by an expression. The expression is evaluated, and the result is printed on the teletypewriter printer.

```
;EXP(SIN(3.4/8))  2.68631        ;system responds on the same line.
```

When the user interrupts a program in execution, he can obtain current values of variables in the program or make calculations using the current values.

```
(ESC)
STOP AT 0500
;A1    -.51125
;ABS(-.511125.5↑.5)  714.927
```

LET Command

The programmer can change the value of an arithmetic variable in the program by issuing the LET command.

```
(ESC)
 STOP AT 1100
 PRINT A ) 0          ;user checks value of variable A.

 LET A = -1 )         ;user changes the value of arithmetic variable A and resumes
                       running at statement 505.
 GOTO 505 )
```

RESTORE Command

The user may wish to restore the data block pointer to the top of the data block . The RESTORE command gives the user this option.

```
(ESC)
 STOP AT 2500
 RESTORE )
 GOTO 2500 )
```

DIM Command

As described in Chapter 2, an array can be redimensioned as long as the number of elements after redimensioning is the same or smaller than the original number declared. The DIM command can be used for this purpose also.

```
20 DIM A(4,4)
        .
        .
        .
(ESC)
STOP AT 500

DIM A(3,5) )
```

APPENDIX A

Single-User BASIC Error Messages

Error messages are printed as two-digit codes, which are defined below.

Error Code	Meaning
00	Format error
01	Illegal character
02	Syntax error
04	System error
05	Illegal statement number
06	Too many variable names
07	Spelling error
08	Spelling error
09	No such word
10	Incorrect subscript closure
11	Incorrect parenthesis closure
12	Not a keyboard command
13	No such line number
14	Storage overflow (while inputting program)
15	Read statement is out of data
16	Arithmetic overflow (number too large)
18	Too many nested GOSUBs
19	Too many RETURNs
20	Too many nested FORs
21	FOR without NEXT
22	NEXT without FOR
23	Out of storage (while assigning variable storage)
24	Array too large
25	Attempt to dimension simple variable
26	Variable name is not dimensionable
28	Redimensioned array is larger than previously defined
29	Expression is too complex
30	Illegal format in defined function
31	Subscript exceeds dimension
32	Undefined user function
33	Too many nested functions
34	Negative subscript
35	Function not yet implemented

APPENDIX B

OPERATING PROCEDURES

Loading Single-user BASIC

Single User BASIC (tape number 091-000018) is loaded by use of the binary loader. Once loaded, BASIC will type:

> DO YOU WISH COMMAS TO BE 14 SPACES (TYPE Y),
> OR 15 (TYPE N) ?

The user can adjust print columns to either 14 spaces (70 for full page) or 15 spaces (75 for full page). Once the query has been answered, BASIC will initialize itself and in that process destroy the binary loader, leaving intact the bootstrap loader.

If memory is larger than 4K it will preserve both the binary and the bootstrap loaders and will use the additional core to store the user's data and program. Larger memory configurations, therefore, provide the user with the capability of handling larger programs with larger bodies of data. After initializing itself, BASIC performs a carriage return/line feed and waits for the user to respond.

In system with 4K of memory, the query:

> DO YOU WISH TO OVERWRITE THE FUNCTIONS SIN, COS,
> ATN, AND TAN (TYPE Y OR N)?

is printed on the teletypewriter. If the user responds with Y, the core required for these functions is made available for storage of the user's program and data, thereby expanding the total storage available to the user by sixty percent.

Restarting Single-user BASIC

If at some point in working with the system the user wishes to restart the program, due to a power failure for example, he may do so by setting the Nova/Supernova panel data switches to the restart address (000002), and pressing RESET and START operating switches.

This action will place the system in idle mode until the user strikes the ESC key on the teletypewriter. All other teletypewriter keys will give no response until the ESC key is struck. When hit, it will cause the system to do a carriage-return on the teletypewriter and type the message "*READY".

Restarting BASIC does not destroy the program or data that the user has entered previously. To accomplish this the user must issue the keyboard command "NEW", which causes the user's program area to be cleared in preparation for a new program.

APPENDIX C

USE OF THE TELETYPE READER/PUNCH

To punch a BASIC program tape with blank leader and trailer, follow this procedure:

1. Turn the power switch on the front of the teletypewriter to the LOCAL position. Press the ON button on top of the teletype punch unit. Then press the HERE IS* key to punch blank tape. When enough blank tape has been produced press the OFF button on top of the punch unit and turn the power switch back to the LINE position.

2. Type the Keyboard command LIST, but do not press the RETURN key yet. Press the ON button on top of the teletype punch unit and then press the RETURN key. After waiting until the punching and listing is complete, press the OFF button on the teletype punch.

3. Repeat step 1 to punch a length of blank trailer on the tape.

4. Remove the tape by pulling straight up. This produces arrows on the ends of the tape to indicate the direction in which the tape should be read through the reader. It is usually wise to write the name of the program on the tape for easy identification.

To load a BASIC program tape, follow this procedure:

1. Set the switch on the teletypewriter's paper tape reader to the STOP position. Release the plastic tape guide on the tape reader and mount the beginning end of the tape in the reader with the arrow at the end of the tape pointing forward and the punched portion hanging down behind. Be sure the sprocket holes in the tape leader are fitted on the sprocket wheel.

2. Close the plastic tape guide and move the switch on the reader to the START position. Wait until the entire tape has been read and listed, and then move the reader switch to the FREE position, and pull out the remaining tape.

---

* On some teletypewriters the HERE IS key may not punch blank tape. In this case, the user should hold down CTRL, SHIFT, REPT, and P keys simultaneously to punch blank tape. The REPT and P keys should be released first.

INDEX

DataGeneral

PROGRAMMING DOCUMENTATION
REMARKS FORM

Document Title	Document No.	Tape No.

SPECIFIC COMMENTS: List specific comments. Reference page numbers when applicable. Label each comment as an addition, deletion, change or error if applicable.

GENERAL COMMENTS: Also, suggestions for improvement of the Publication.

FROM:

Name	Title	Date

Company Name

Address (No. & Street)	City	State	Zip Code

Form No. 10-24-004

FOLD DOWN FIRST FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary If Mailed In The United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Programming Documentation

FOLD UP SECOND FOLD UP

STAPLE