Digital Equipment Corporation
Maynard, Massachusetts

digital

BASIC-8

TIME SHARING SYSTEM
TSS/8

DEC-T8-KJZA-D

BASIC-8
PROGRAMMING MANUAL

A manual for BASIC-8, the elementary algebraic language designed for use with the PDP-8/I Time-Sharing System. This manual is preliminary, and subject to change without notice.

For additional copies of this document order No. DEC-T8-KJZA-D from Program Library, Digital Equipment Corporation, Maynard, Mass. Price $2.50

DIGITAL EQUIPMENT CORPORATION ▫ MAYNARD, MASSACHUSETTS

1st Printing March 1969

Your attention is invited to the last two pages of this manual. The How To Obtain Revisions and Corrections offers you a means of keeping up-to-date with DEC's software. The Reader's Comments Card, when filled in and returned, is beneficial to both you and DEC. Each card received is considered when documenting subsequent manuals, and where the comments imply or ask for assistance, a knowledgeable DEC representative will contact you.

BASIC® is a conversational algebraic language, originally developed at Dartmouth College with support from a grant by the National Science Foundation. Digital Equipment Corporation is grateful to the Trustees of Dartmouth College for permission to reprint this manual, and also wishes to acknowledge its appreciation to the project directors, Dr. John G. Kemeny, professor of mathematics, and Dr. Thomas E. Kurtz, director of the Kiewit Computation Center at Dartmouth College.

CHAPTER 1
INTRODUCTION

BASIC-8 (hereafter referred to as BASIC) is designed to run under control of the PDP-8/I Time-Sharing System (TSS/8) and allows many users to have equal access to the computer, keeping records of the time used by each user.

Learning to use the system is quite easy. After you have been assigned a Teletype console, you begin by pressing the RETURN key. This tells TSS/8 that you are ready, and it responds by typing a period.

You are now ready to log in to the system. Following the period you type

```
.LOGIN (number) (password)
```

where the number is your account (or project) number, and the password is the code name assigned to you. For instance, you might actually type

```
.LOGIN 6 BILL
```

and terminate this line by typing the RETURN key. If the number and name that you have typed are acceptable, TSS/8 types another period, and you respond by typing DUPLEX which is the Teletype mode in which you will operate.

Since you are going to write a program in BASIC language, you must call the BASIC program by typing

```
.R BASIC
```

You are now ready to begin programming in BASIC, but first you must learn the BASIC language, and that's what this manual is all about.

BASIC is intended for use by students, and also by scientists and engineers, who wish to run algebraic problems on the computer with a minimum of learning time. Program statements are written in English, and mathematical expressions are written in standard notation, with a few substitutions (see Section 2.2). The asterisk (*) is used as a multiplication sign, the slash (/) for division and the up arrow (↑) for raising to a power. To write A times B, divided by X squared, type

```
A*B/X↑2
```

The language is fairly simple but it must be written very exactly. The computer must be able to interpret your BASIC program statements, without ambiguity, so you must follow carefully the instructions for writing statements.

A word about programming. A program is a sequence of statements or a set of instructions, which you give to BASIC to perform an operation, or to solve a certain problem. Normally, BASIC will execute your first statement, then the second, and so on to the last. BASIC is particularly suited to repetitive calculations, therefore, several techniques may be used to repeat the execution of parts of your program. This process of repeating is called looping.

If you write a statement incorrectly, BASIC will type an error message to help you identify and correct the error. Some of these error messages will be given in a numbered code as presented in the Error Message list in Appendix C. Other messages are printed out in English. But don't expect BASIC to make corrections if the results are bad; there is probably something wrong in the program, and you will have to find and correct this yourself.

The TSS/8 BASIC system was designed to provide computer service for many users with relatively small programs, however, there are no general restrictions on program length. If your program becomes too large to store in the computer and execute at one time, the system will store part of your program on the disk, and run each part separately. This process is called automatic segmentation, and is one of the advanced features of the TSS/8 BASIC system.

With a little practice at the Teletype console, you will soon be writing BASIC programs on the PDP-8/I.

For more information about TSS/8, see "Time-Sharing System TSS/8 Monitor," Order No. DEC-T8-MRFA-D.

CHAPTER 2

THE BASIC LANGUAGE

2.1 AN EXAMPLE

The following example is a complete BASIC program for solving a system of two simultaneous linear equations in two variables:

$$ax + by = c$$
$$dx + ey = f$$

and then solving two different systems, each differing from this system only in the constants c and f.

You should be able to solve this system, if ae - bd is not equal to 0, to find that

$$x = \frac{ce - bf}{ae - bd} \qquad \text{and} \qquad y = \frac{af - cd}{ae - bd}.$$

If ae - bd = 0, there is either no solution or there are infinitely many, but there is no unique solution. If you are rusty on solving such systems, take our word for it that this is correct. For now, we want you to understand the BASIC program for solving this system.

Study this example carefully; in most cases the purpose of each line in the program is self-evident, then read the commentary and explanation.

```
10 READ A, B, D, E
15 LET G=A*E-B*D
20 IF G=0 THEN 65
30 READ C, F
37 LET X=(C*E-B*F)/G
42 LET Y=(A*F-C*D)/G
55 PRINT X, Y
60 GO TO 30
65 PRINT "NO UNIQUE SOLUTION"
70 DATA 1, 2, 4
80 DATA 2, -7, 5
85 DATA 1, 3, 4, -7
90 END
```

Immediately we observe several things about this sample program. First, we see that the program uses only capital letters, since the Teletype has only capital letters. We also see that the letter "oh" is distinguished from the numeral "zero" by having a diagonal slash through the "zero". We make the distinction since, in a computer program, it is not always possible to tell from the context whether the letter or the numeral was intended, unless they have a different appearance. This distinction is

made automatically while typing, since the Teletype has one key for "oh" and another for "zero"; and one key for "one", another for the letter "i", and no key for the lower case letter "l".

A second observation is that each line of the program begins with a number. These numbers are called line numbers (and may range from 1 through 2046), and serve to identify the lines, each of which is called a statement. Thus, a program is made up of statements, most of which are instructions to BASIC. Line numbers also serve to specify the order in which the statements are to be performed by BASIC, therefore you may type your line numbers in any order, however, best results will be obtained if they are in ascending order. As you type, BASIC sorts out and edits the program, putting the statements into the order specified by their line numbers. (This editing process facilitates the correcting and changing of programs, as explained later.)

A third observation shows that each statement starts, after its line number, with an English word which denotes the type of the statement. There are several types of statements in BASIC, nine of which are discussed in this chapter. Seven of these nine appear in the sample program, above.

A fourth observation, not at all obvious from the program, is that spaces have no significance in BASIC, except in messages which are to be printed out, as in line number 65 above. Thus, spaces may be used or not used to improve the appearance of a program and make it more readable. Statement 10 could have been typed as 10READA,B,D,E and statement 15 as 15LETG=A*E-B*D.

With this preface, let us go through the example, step by step. The first statement, 10, is a READ statement. It must be accompanied by one or more DATA statements. When BASIC encounters a READ statement while executing your program, it will cause the variables (A,B,D,E) listed after the READ to be given values according to the next available numbers in the DATA statements (lines 70, 80, and 85). In the example, we read A in statement 10 and assign the value 1 to it from statement 70, similarly with B and 2, and with D and 4. At this point, we have exhausted the available data in statement 70, but there is more in statement 80, so we pick up from it the number 2 to be assigned to E.

Next we go to statement 15, which is a LET statement, and encounter a formula to be evaluated. (The asterisk "*" is used to denote multiplication.) In this statement, we direct BASIC to compute the value of AE - BD, and to call the result G. In general, a LET statement directs BASIC to set a variable equal to the formula on the right side of the equal sign.

We know that if G is equal to zero, the system has no unique solution, therefore, we ask in line 20, if G is equal to zero. If BASIC discovers a yes answer to the question, it is directed to go to line 65, where it prints NO UNIQUE SOLUTION. From this point, it would go to the next statement, but lines 70, 80, and 85 give it no instructions since DATA statements are not executed, therefore, it goes to line 90 which tells it to END the program.

If the answer to the question "Is G equal to zero?" is no, as it is in this example, BASIC goes on to the next statement, in this case 30. (Thus, an IF-THEN tells BASIC where to go if the IF

condition is met, or to go on to the next statement if it is not met.) BASIC is now directed to read the next two entries from the DATA statements, -7 and 5, (both are in statement 80) and to assign them to C and F respectively. BASIC is now ready to solve the system

$$x + 2y = -7$$
$$4x + 2y = 5$$

In statements 37 and 42, we direct BASIC to compute the value of X and Y according to the formulas provided. Note that we must use parentheses to indicate that CE - BF is divided by G; without parentheses, only BF would be divided by G, which would let $X = CE - \frac{BF}{G}$.

BASIC is told to print the two values computed, that of X and that of Y, in line 55, then it moves on to line 60 where it is directed back to line 30. If there are additional numbers in the DATA statements, as there are here in 85, it is told in line 30 to take the next number and assign it to C, and the one after that to F. BASIC is now ready to solve the system

$$x + 2y = 1$$
$$4x + 2y = 3$$

As before, it finds the solution in 37 and 42 and prints them out in 55, and then is directed in 60 to go back to 30.

In line 30 BASIC reads two more values, 4 and -7, which are found in line 85, and then proceeds to solve the system

$$x + 2y = 4$$
$$4x + 2y = -7$$

and to print out the solutions. It is directed back to 30, but there are no more pairs of numbers available for C and F in the DATA statements. BASIC then informs you that it is out of data by typing OUT OF DATA and stops.

Let us look at the importance of the various statements. For example, what would have happened if we had omitted line number 55? The answer is simple; BASIC would have solved the three systems and then told us when it was out of data. However, since it was not asked to tell us (PRINT) its answers, the solutions would be BASIC's secret. What would have happened if we had left out line 20? In the problem just solved nothing would have happened, but if G were equal to zero, we would have given BASIC the impossible task of dividing by zero in 37 and 42, and it would tell us so by printing

1Ø IN 37 and 1Ø IN 42. If we left out statement 60, BASIC would have solved the first system, printed out the values of X and Y, and then gone on to line 65 where it would be directed to print NO UNIQUE SOLUTION. It would do this and then stop.

One very natural question arises from the seemingly arbitrary numbering of the statements: Why this selection of line numbers? The answer is that the particular choice of line numbers is arbitrary, as long as the statements are numbered in the order in which we want BASIC to follow in executing the program. We could have numbered the statements 1, 2, 3, ..., 13, although we do not recommend this numbering. We would normally number the statements 10, 20, 30, ..., 130, allowing additional statements to be inserted later. Thus, if we find that we have left out two statements between those numbered 40 and 50, we can give then any two numbers between 40 and 50, and in the editing and sorting process, BASIC will put them in their proper place.

Another question arises from the seemingly arbitrary placing of the elements of data in the DATA statements: Why place them as they have been in the example program? Here again the choice is arbitrary and we need only put the numbers in the order that we want them read (the first for A, the second for B, the third for D, the fourth for E, the fifth for C, the sixth for F, the seventh for the next C, etc.) In place of the three statements numbered 70, 80, and 85, we could have put

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

or we could have written, perhaps more naturally,

```
70 DATA 1, 2, 4, 2
75 DATA -7, 5
80 DATA 1, 3
85 DATA 4, -7
```

to indicate that the coefficients appear in the first data statement and the various pairs of right-hand constants appear in the subsequent statements.

The program and the resulting run is shown below exactly as it appears on the Teletype.

```
LIST
  10    READ A, B, D, E
  15    LET G=A*E-B*D
  20    IF G=0 THEN 65
  30    READ C, F
  37    LET X=(C*E-B*F)/G
  42    LET Y=(A*F-C*D)/G
  55    PRINT X, Y
  60    GO TO 30
  65    PRINT "NO UNIQUE SOLUTION"
  70    DATA 1, 2, 4
```

```
  80    DATA 2, -7, 5
  85    DATA 1, 3, 4, -7
  90    END

RUN

 4              -5.5
 .6666667        .1666667
-3.666667        3.833333

1030 ERROR IN LINE    30
OUT OF DATA
READY
```

NOTE

The number 1030 (3rd line from the bottom) in the example above is an internal code meaning OUT OF DATA. This number may not appear in future versions.

READY, the last line in the printout above, is explained in Chapter 4.

After typing the program, we type RUN followed by a carriage return. Up to this point BASIC stores the program and does nothing with it. It is the RUN command which directs BASIC to execute your program.

The message OUT OF DATA here may be ignored since it means your program has made an attempt to read more data than you have made available in DATA statements.

2.2 ARITHMETIC OPERATIONS

BASIC can add, subtract, multiply, divide, extract square roots, raise a number to a power, and find trigonometric functions such as sine and cosine. We shall now learn how to tell BASIC to perform these various operations in the order that we want them done.

BASIC performs its primary function (that of computation) by evaluating formulas which are supplied in a program. These formulas are very similar to those used in standard mathematical calculation, with the exception that all BASIC formulas must be written on a single line. Five arithmetic operators can be used to write a formula:

Symbol	Example	Meaning
+	A + B	Addition (add B to A)
-	A - B	Subtraction (subtract B from A)
*	A * B	Multiplication (multiply B by A)
/	A / B	Division (divide A by B)
↑	X ↑ 2	Raise to the power (find X^2)

We must be careful with parentheses to make sure that those things which we want together are grouped together. We must also understand the order in which BASIC operates. For example, if we type A + B * C ↑ D, BASIC will first raise C to the power D, multiply this result by B, and then add A to the resulting product. This is the same convention as is usual for: $A + B \times C^D$. If this is not the order intended, then we must use parentheses to indicate a different order. For example, if it is the product of B and C that we want raised to the power D, we must write A + (B * C) ↑ D; or, if we want to multiply A + B by C to the power D, we write (A + B) * C↑D. We could even add A to B, multiply the sum by C, and raise the product to the power D by writing ((A + B) * C) ↑D. The order of priorities is summarized in the following rules:

1. The formula inside parentheses is computed before the parenthesized quantity is used in further computations.

2. In the absence of parentheses in a formula involving addition, multiplication, and the raising of a number to the power, BASIC first performs exponentiation, then performs the multiplication, and the addition comes last. Division has the same priority as multiplication, and subtraction the same as addition.

3. In the absence of parentheses in a formula involving only multiplication and division, the operations are performed from left to right, just as they are read. Addition and subtraction is performed from left to right also.

These rules which are illustrated in the previous example, tell us that when BASIC is faced with A - B - C, it will (as usual) subtract B from A and then C from the difference; with A/B/C, it will divide A by B and that quotient by C; with A↑B↑C, it will raise the number A to the power B and take the resulting number and raise it to the power C. To avoid a question of priority, you may put parentheses in as necessary to eliminate possible ambiguities.

2.2.1 Functions

In addition to the five arithmetic operators, BASIC can evaluate several mathematical functions. These functions are given special three-letter names, as the following list shows:

Functions	Interpretation	
SIN (X)	Find the sine of X	X interpreted as a number, or as an angle measured in radians.
COS (X)	Find the cosine of X	
TAN (X)	Find the tangent of X	
ATN (X)	Find the arctangent of X	
EXP (X)	Find e^X (2.712818)	
LOG (X)	Find a natural logarithm of X ($\log_e X$)	
ABS (X)	Find the absolute value of X (|X|)	
SQR (X)	Find the square root of X ($\sqrt{X}$)	

Three other functions are also available in BASIC: INT, RND, and SGN; these are reserved for explanation in Section 3.2.

In place of X, we may substitute any formula or any number in parentheses following any of these functions. For example, BASIC may be asked to find $\sqrt{4 + X^3}$ by writing SQR (4 + X↑3), or the arc-tangent of $3X - 2e^X + 8$ by writing

```
ATN (3*X-2* EXP(X)+8)
```

If the value of $(\frac{5}{6})^{17}$ is needed, you can write the two line program

```
10 PRINT (5/6)↑17
20 END

RUN

 .04503441

READY
```

and BASIC will find the decimal form of this expression and print it out in less time than it took to type either line.

2.2.2 Numbers and Variables

Since we have mentioned numbers and variables, it should be understood how to write numbers for BASIC and what variables are allowed. A number may be positive or negative and may contain up to 8-1/3 significant digits (absolute limit is ±134217727) but it must be expressed in decimal form. For example, all of the following are numbers in BASIC: 2, -3.675, 123456789, -.98765432, and 483.4156. The following are not numbers in BASIC: 14/3, $\sqrt{7}$, and .001234567890. The first two are formulas but not numbers, and the last one has more than 8-1/3 significant digits. BASIC may be asked to find the decimal expansion of 14/3 or $\sqrt{7}$, and to do something with the resulting number, but neither may be included in a list of DATA. Further flexibility is gained by use of the letter E (exponent), which stands for "times ten to the power"; thus, .00123456789 may be written in any of several forms: .123456789E-2 or 123456789E-11 or 1234.56789E-6. Ten million may be written as 1E7 (or 1E + 7) and 1969 as 1.969E3 (or 1.969E + 3). E7 is not written as a number, but as 1E7 to indicate that it is 1 that is multiplied by 10^7. Numbers must be in the range .14E-38<N<1.7E38.

The BASIC program performs computations in this E (or floating-point) format. Results are printed out in decimal format for numbers in the range 0.01<N<100000. Trailing decimal points are omitted. Leading and trailing zeroes are also omitted, except when the value is zero or when the number is in the range 0.01≤N<0.1. Numbers outside the range 0.01≤N<100000 are printed out in E format.

A numerical variable in BASIC is denoted by any letter, or by any letter followed by a single digit. The computer therefore will interpret E7 as a variable, along with A, X, N5, J0, and M1. A variable in BASIC stands for a number, usually one that is not known to the programmer at the time the program was written. Variables are given or assigned values by LET and READ statements. The value so assigned will not change until the next time a LET or READ statement is encountered with a value for that variable. However, all variables are set equal to zero before a RUN; thus, it is only necessary to assign a value to a variable when a value other than zero is required.

Although BASIC does little in the way of correcting, during computation it will sometimes help you when you forget to indicate absolute value. For example, if BASIC is asked for the square root of -7 or the logarithm of -5, it will give the square root of 7 with the error message for the square root of a negative number, or the logarithm of 5 with the error message for the logarithm of a negative number.

2.2.3 Symbols of Relation

Six other mathematical symbols, symbols of relation, are used in BASIC, and these are used in IF-THEN statements where it is necessary to compare values. An example of the use of these relation symbols was given in the example program in Section 2.1.

Any of the following six standard relations may be used:

Symbol	Example	Meaning
=	A = B	Is equal to (A is equal to B)
<	A < B	Is less than (A is less than B)
<=	A <= B	Is less than or equal to (A is less than or equal to B)
>	A > B	Is greater than (A is greater than B)
>=	A >= B	Is greater than or equal to (A is greater than or equal to B)
<>	A <> B	Is not equal to (A is not equal to B)

2.3 LOOPS

We are frequently interested in writing a program in which one or more portions are performed not just once but a number of times, perhaps with slight changes each time. To write the simplest program, the one in which the portion to be repeated is written just once, we use the programming device known as a loop.

Programs which use loops can be best illustrated and explained by two programs which print out a table of the first 100 positive integers together with the square root of each. Without a loop, the program would be 101 lines long and read:

```
10 PRINT 1, SQR(1)
20 PRINT 2, SQR(2)
30 PRINT 3, SQR(3)
      .
      .
      .
990 PRINT 99, SQR(99)
1000 PRINT 100, SQR(100)
1010 END
```

With the following program, using one type of loop, we can obtain the same table with 5 lines instead of 101:

```
10 LET X=1
20 PRINT X, SQR(X)
30 LET X=X + 1
40 IF X<=100 THEN 20
50 END
```

Statement 10 gives the value of 1 to X and "initializes" the loop. In line 20, both 1 and its square root are printed. Then, in line 30, X is increased by 1, to 2. Line 40 asks whether X is less than or equal to 100; an affirmative answer directs BASIC back to line 20. Here BASIC prints 2 and $\sqrt{2}$, and goes to 30. Again X is increased by 1; this time to 3, and at 40 it goes back to 20. This process is repeated (line 20 (print 3 and $\sqrt{3}$), line 30 (X = 4), line 40 (since 4 < 100 go back to line 20), etc.) until the loop has been traversed 100 times. Then, after it has printed 100 and its square root, X becomes 101. BASIC now receives a negative answer to the question in line 40 (X is greater than 100, not less than or equal to it), it does not return to 20 but moves on to line 50, and ends the program. All loops contain four characteristics:

1. initialization (line 10),
2. the body (line 20),
3. modification (line 30), and
4. an exit test (line 40).

Because loops are so important and because loops of the type just illustrated arise so often, BASIC provides two statements to specify a loop even more simply. They are the FOR and NEXT statements, and their use is illustrated in the program:

```
10 FOR X=1 TO 100
20 PRINT X, SQR(X)
30 NEXT X
50 END
```

In line 10, X is set equal to 1, and a test is set up, like that of line 40 in the previous example program. Line 30 carries out two tasks: X is increased by 1, and the test is made to determine whether to go back to 20 or go on. Thus lines 10 and 30 take the place of lines 10, 30, and 40 in the previous program, and they are easier to use.

Note that the value of X is increased by 1 each time we go through the loop. If we wanted a different increase, as in increments of 5, we could specify it by writing

```
10 FOR X=1 TO 100 STEP 5
```

and BASIC would assign 1 to X on the first time through the loop, 6 to X on the second time through, 11 on the third time, and 96 on the 20th and last time through the loop. (Another step of 5 would take X beyond 100.) The program would proceed to the end after printing 96 and its square root. The STEP may be positive or negative, and we could have obtained the first table, printed in reverse order, by writing line 10 as

```
10 FOR X=100 TO 1 STEP -1
```

In the absence of a STEP instruction, a step size of +1 is assumed.

More complicated FOR statements are allowed. The initial value, the final value, and the step size may all be formulas of any complexity. For example, if N and Z have been specified earlier in the program, we could write

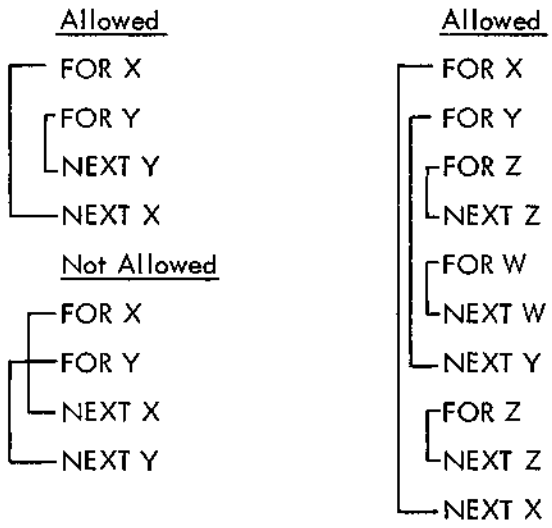
```
FOR X=N + 7 * Z TO (Z-N)/3 STEP (N-4*Z)/10
```

For a positive step-size, the loop continues as long as the control variable is less than or equal to the final value. For a negative step-size, the loop continues as long as the control variable is greater than or equal to the final value.

If the initial value is greater than the final value (less than, for negative step-size), then the body of the loop will not be performed at all, instead, BASIC will immediately pass to the statement following the NEXT. As an example, the following program for adding up the first n integers will give the correct result 0 when n is 0.

```
10 READ N
20 LET S=0
30 FOR K=1 TO N
40 LET S=S + K
50 NEXT K
60 PRINT S
70 GO TO 10
90 DATA 3, 10, 0
99 END
```

It is often useful to have loops within loops, called nested loops, which can be expressed with FOR and NEXT statements, however, they must actually be nested and must not cross, as the following skeleton examples illustrate:



2.4 LISTS AND TABLES

In addition to the ordinary variables used by BASIC, there are variables which can be used to designate the elements of a list or a table. These are used where we might ordinarily use a subscript or a double subscript; for example, the coefficients of a polynomial (a_0, a_1, a_2,...) or the elements of a matrix ($b_{i,j}$). The variables which we use in BASIC consists of a single letter, which we call the name of the list, followed by the subscripts in parentheses. Thus, we might write A(0), A(1), A(2), etc., for the elements of the matrix.

When using subscripts, a DIMENSION (DIM) statement must be used to indicate that BASIC must save extra space for the list or table. The dimension statement consists of the command DIM, a space, and the variable followed by the largest subscript (parenthesized) to be assigned. As shown in the following examples, more than one variable may be declared in a single DIM statement.

The list A(0), A(1),...,A(10) may be entered into a program very simply by the lines:

```
Ø5 DIM A(1Ø)
1Ø FOR I=Ø TO 1Ø
2Ø READ A(I)
3Ø NEXT I
4Ø DATA 2, 3, -5, 5, 2.2, 4, -9, 123, 4, -4, 3
```

We can enter a 3x5 table into a program by writing:

```
05 DIM B(2,4)
10 FOR H=0 TO 2
20 FOR J=0 TO 4
30 READ B(H,J)
40 NEXT J
50 NEXT H
60 DATA 2, 3, -5, -9, 2
70 DATA 4, -7, 3, 4, -2
80 DATA 3, -3, 5, 7, 8
```

The single letter denoting a list or a table name may also be used to denote a simple variable without confusion. However, the same letter may not be used to denote both a list and a table in the same program. The form of the subscript is quite flexible, and you might have the list item B(H+K) or the table items B(H,K) or Q(A(3,7), B-C).

A list and a run of a problem which uses both a list and a table is shown below. The program computes the total sales of each of five salesmen, all of whom sell the same three products. The list P gives the price/item of the three products and the table S tells how many items of each product each man sold. The program indicates that product No. 1 sells for $1.25 per item, No. 2 for $4.30 per item, and No. 3 for $2.50 per item; and also that salesman No. 1 sold 40 items of the first product, 10 of the second, and 35 of the third, and so on. The program reads in the price list in lines 40-80, using data in lines 910-930. The same program could be used again, modifying only line 900 if the prices change, and only lines 910-930 to enter the sales in another month.

Since the DIM statement is not executed, it may be entered into the program on any line before END and prior to use of the list or table; it is convenient, however, to place DIM statements near the beginning of the program.

```
  5  DIM P(3), S(3,5)
 10  FOR I = 1 TO 3
 20    READ P(I)
 30  NEXT I
 40  FOR I = 1 TO 3
 50    FOR J = 1 TO 5
 60      READ S(I,J)
 70  NEXT J
 80  NEXT I
 90  FOR J = 1 TO 5
100    LET S = 0
110    FOR I = 1 TO 3
120      LET S = S + P(I)*S(I,J)
130    NEXT I
140    PRINT "TOTAL SALES FOR SALESMAN "J, "$" S
150 NEXT J
900 DATA 1.25, 4.30, 2.50
910 DATA 40, 20, 37, 29, 42
```

```
920 DATA 10, 16, 3, 21, 8
930 DATA 35, 47, 29, 16, 33
990 END

RUN

TOTAL SALES FOR SALESMAN 1   $ 180.5
TOTAL SALES FOR SALESMAN 2   $ 211.3
TOTAL SALES FOR SALESMAN 3   $ 131.65
TOTAL SALES FOR SALESMAN 4   $ 166.55
TOTAL SALES FOR SALESMAN 5   $ 169.4
```

2.5 ELEMENTARY BASIC STATEMENTS

This section contains a short and concise description of each type of BASIC statement discussed earlier in this chapter and adds one statement to the list. In each form, a line number is assumed, and brackets denote a general type, thus, [variable] refers to any variable, which is a single letter, possibly followed by a single digit.

2.5.1 LET

This statement is not a statement of algebraic equality, but rather a command to BASIC to perform certain computations and to assign the answer to a certain variable. Each LET statement is of the form:

LET [variable] = [formula].

For example:

```
100 LET X=X + 1
259 LET W7=(W-X4↑3) * (Z-A/(A-B)) - 17
```

2.5.2 READ and DATA

A READ statement is used to assign to the listed variables, values obtained from a DATA statement. Neither statement is used without the other type. A READ statement causes the variables listed in it to be given, in order, the next available numbers in the collection of DATA statements. Before the program is run, BASIC takes all of the DATA statements in the order in which they appear and creates a large data block. Each time a READ statement is encountered anywhere in the program, the data block supplies the next available number or numbers. If the data block runs out of data with a READ statement still asking for more, the program is assumed to be done and an OUT OF DATA message is received.

Since data must be read in before it can be worked with, READ statements normally occur near the beginning of a program. The location of DATA statements is arbitrary, as long as they occur in the correct order. A common practice is to collect all DATA statements and place them just before the END statement.

Each READ statement is of the form:

READ [sequence of variables]

and each DATA statement is of the form:

DATA [sequence of numbers]

Examples:

```
150 READ X, Y, Z, X1, Y2, Q9
330 DATA 4, 2, 1.7
340 DATA 6.734E-3, -174.321, 3.1415926

234 READ B(K)
263 DATA 2, 3, 5, 7, 9, 11, 10, 8, 6, 4

10 READ R(I,J)
440 DATA -3, 5, -9, 2.37, 2.9876, -437.234E-5
450 DATA 2.765, 5.5576, 2.3789E2
```

Remember that only numbers are put in a DATA statement, and that 15/7 and $\sqrt{3}$ are formulas, not numbers.

2.5.3 PRINT

The PRINT statement has a number of different uses, which are discussed in more detail in Chapter 3. The common uses are described below.

1. To print out the result of some computation:

```
100 PRINT X, SQR(X)
135 PRINT X, Y, Z, B*B-4*A*C, EXP(A-B)
```

The first will print X and then several spaces to the right of that number (X), it's square root. The second will print five different numbers:

$$X,\ Y,\ Z,\ B^2 - 4AC,\ \text{and}\ e^{A-B}$$

BASIC will compute the two formulas and print them, as long as values have been given to A, B, and C. It can print up to five numbers per line in this format.

2. To print out verbatim a message included in the program:

```
100 PRINT "NO UNIQUE SOLUTION"
430 PRINT "X VALUE", "SINE", "RESOLUTION"
```

Both have been encountered in the example programs. The first prints the simple statement; the second prints the three labels with spaces between them. The labels in 430 automatically line up with three numbers called for a PRINT statement.

3. A combination of 1. and 2. above:

```
150 PRINT "THE VALUE OF X IS" X
 30 PRINT "THE SQUARE ROOT OF" X, "IS" SQR(X)
```

If the first has computed the value of X to be 3, it will print out:

```
THE VALUE OF X IS 3
```

If the second has computed the value of X to be 625, it will print out:

```
THE SQUARE ROOT OF 625 IS 25
```

4. To skip a line (explained in Chapter 3).

We have seen examples of the first three in our previous example programs. Each type is slightly different in form, but all start with PRINT after the line number.

2.5.4 GO TO

In a program there are times when you do not want all commands executed in the order that they appear in the program. If we do not want the program to go to the END statement yet, but to go through the same process for a different value, we direct BASIC to go back to a certain line with a GO TO statement; in the form:

GO TO [line number]

Example:

```
150 GO TO 75
```

2.5.5 IF -- THEN

There are times when we are interested in jumping the normal sequence of commands, if a certain relationship holds. For this we use an IF--THEN statement, sometimes called a conditional GO TO statement. Each such statement is of the form:

IF [formula] [relation] [formula] THEN [line number]

Examples:

```
40 IF SIN(X) <= M THEN 80
20 IF G = 0 THEN 65
```

The first asks if the sine of X is less than or equal to M, and directs the computer to skip to line 80 if it is. The second asks if G is equal to 0, and directs the computer to skip to line 65 if it is. In each case, if the answer to the question is no, BASIC will go to the next line of the program.

2.5.6 IF...GO TO

To give a different structure to the IF-THEN instructions, the instruction

```
IF X > 5 THEN 200
```

may also be written as

```
IF X > 5 GO TO 200
```

2.5.7 FOR and NEXT

We have already encountered the FOR and NEXT statements in our loops, and have seen that they go together, one at the entrance to the loop and one at the exit, directing BASIC back to the entrance again. Every FOR statement is of the form:

FOR [variable] = [formula] TO [formula] STEP [formula]

Most commonly, the expressions will be integers and the STEP omitted. In the latter case, a step size of one is assumed. The accompanying NEXT statement is simple in form, but the variable must be precisely the same one as that following FOR in the FOR statement. Its form is NEXT [variable]. The variable used in the FOR and NEXT statements may not be subscripted.

Examples:

```
 30   FOR X=0 TO 3 STEP D
 80   NEXT X
120   FOR X4 = (17+COS(Z))/3 TO 3*SQR(10) STEP 1/4
235   NEXT X4
240   FOR X=8 TO 3 STEP -1
456   FOR J=-3 TO 12 STEP 2
```

Notice that the step size may be a formula (1/4), a negative number (-1), or a positive number (2). In the example with lines 120 and 235, the successive values of X4 will be .25 apart, in increasing order. In line 240, the successive values of X will be 8, 7, 6, 5, 4, 3. In line 456, on successive trips through the loop, J will take on values -3, -1, 1, 3, 5, 7, 9, and 11.

If the initial, final, or step-size values are given as formulas, these formulas are evaluated once and for all upon entering the FOR statement. The control variable can be changed in the body of the loop; of course, the exit test always uses the latest value of this variable.

If you write 50 FOR Z = 2 TO -2, without a negative step size, the body of the loop will not be performed and BASIC will proceed to the statement immediately following the corresponding NEXT statement.

2.5.8 DIM

Whenever we want to enter a list or a table, we must use a DIM statement to inform BASIC to save sufficient room for the list or table. Examples:

```
20 DIM H(35)
35 DIM Q(5,25)
```

An alternate way of writing this would be:

```
20 DIM H(35), Q(5,25)
```

The first would enable us to enter a list of 35 items, or 36 if we use H(0); and the latter a table 5 x 25, or by using row 0 and column 0 we get a 6 x 26 table. The DIM statement must precede any other references to the dimensioned variable named.

2.5.9 END

Every program must have an END statement, and it must be the statement with the highest line number in the program. Its form is simple: a line number with END.

```
999 END
```

2.6 ERRORS AND DEBUGGING

It may occasionally happen that the first run of a new problem will be free of errors and give the correct answers, but it is much more common that errors will be present and will have to be corrected. Errors are of two types: errors of form (or syntactical errors) which prevent the running of the program; and logical errors (bugs) in the program which cause BASIC to produce wrong answers or no answers at all.

Errors of form will cause error messages to be printed, and the various types of error messages are listed and explained in Appendix C. Logical errors are often much harder to uncover, particularly when the program gives answers which seem to be nearly correct. In either case, after the errors are discovered, they can be corrected by changing lines, by inserting new lines, or by deleting lines from the program. As indicated in the last section, a line is changed by typing it correctly with the same line number; a line is inserted by typing it with a line number between those of two existing lines; and a line is deleted by typing its line number and pressing the RETURN key. Notice that you can insert a line only if the original line numbers are not consecutive integers. For this reason, most programmers will start out using line numbers that are multiples of five or ten, but that is a matter of choice.

These corrections can be made at any time, when in the editing phase and either before or after a run, by simply retyping the offending line with its original line number.

CHAPTER 3
ADVANCED BASIC

3.1 MORE ABOUT PRINT

The uses of the PRINT statement were described in 2.5.3, but more detail is presented in this chapter. Although the format of answers is automatically supplied for the beginner, the PRINT statement permits a greater flexibility for the more advanced programmer who wishes a different format for his output.

The Teletype line is divided into five zones of fourteen spaces each. Some control of the use of these comes from the use of the comma: a comma is a signal to move to the next print zone or, if the fifth print zone has just been filled, to move to the first print zone of the next line.

For example, if you were to type the program

```
10   FOR N=1 TO 15
20   PRINT N
30   NEXT N
40   END
```

BASIC would print 1 at the beginning of a line, 2 at the beginning of the next line, and so on, finally printing 15 on the fifteenth line. But, by adding a comma to line 20 to read

```
20 PRINT N,
```

you would have the numbers printed in the zones, reading

```
 1             2             3             4             5
 6             7             8             9             10
 11            12            13            14            15

READY
```

More compact output can be obtained by use of the semicolon. If a label (expression in quotes) is followed by a semicolon, the label is printed with no space after it. If a variable is followed by a semicolon, its value is printed in the following format:

First, a minus sign for negative numbers or a space for positive numbers,

then, the numerical value,

then, a single space.

Thus, printing a list of numbers in semicolon format will pack them in the closest readable form.

If you wanted the numbers printed in this fashion, but more tightly packed, you would change line 20 to replace the comma by a semicolon:

```
20 PRINT N;
```

and the result would be printed

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

You should remember that a label inside quotation marks is printed just as it appears and also that the end of a PRINT signals a new line, unless a comma or semicolon is the last symbol.

Thus, the instruction

```
50 PRINT X, Y
```

will result in the printing of two numbers and the return to the next line, while

```
50 PRINT X, Y,
```

will result in the printing of these two values and no return. The next number to be printed will occur in the third zone, after the values of X and Y in the first two.

Since the end of a PRINT statement signals a new line,

```
250 PRINT
```

will cause BASIC to advance the Teletype paper one line. It will put a blank line in your program, if you want to use it for vertical spacing of your results, or it causes the completion of a partially filled as illustrated in the following fragment of a program:

```
50  FOR M = 1 TO N
110 FOR J = 0 TO M
120 PRINT B(M,J);
130 NEXT J
140 PRINT
150 NEXT M
```

This program will print B(1,0) and next to it B(1,1). Without line 140, BASIC would then go on printing B(2,0), B(2,1), and B(2,2) on the same line, and then B(3,0), B(3.1), etc. Line 140 direct BASIC, after printing the B(1,1) value corresponding to M = 1, to start a new line and to do the same thing after printing the value of B(2,2) corresponding to M = 2, etc.

The instructions

```
50 PRINT "TIME-"; "SHAR"; "ING";
51 PRINT " AT"; " DEC"
52 END
```

will result in the printing of

```
TIME-SHARING AT DEC
```

The following rules for the printing of numbers will help you in interpreting your printed results:

1. If a number is an integer, the decimal point is not printed. If the integer contains more then five digits, the number will be printed in E format; the Teletype will give you the first digit, followed by (a) a decimal point, (b) the next six digits, and (c) an E, followed by the appropriate signed integer. For example, it will take 32,437,580 and write it as 3.243758E+07.

2. For any decimal number, no more than seven significant digits are printed.

3. For a number less than 0.01, the E notation is used.

4. Trailing zeroes after the decimal point are not printed. The following program, in which we print out powers of 2, shows how numbers are printed.

```
10 FOR N = -5 TO 16
20 PRINT 2↑N;
30 NEXT N
40 END

RUN

0.03125 0.0625 0.125 0.25 0.5 1 2 4 8 16 32 64 128 256 512
1024 2048 4096 8192 16384 32768 65536
```

3.2 FUNCTIONS

Three functions were mentioned in Section 2.2; they are described below.

3.2.1 The INT Function

The INT function frequently appears in algebraic computation as X, and it gives the greatest integer not greater than X. Thus, INT (2.35) = 2, INT (-2.35) = -2, and INT (12) = 12.

One use of the INT function is to round numbers. We may use it to round to the nearest integer by asking for INT (X + .5). This will round 2.9, for example, to 3, by finding INT (2.9 + .5) = INT (10*X + .5)/10↑2 will round X correct to two decimal places, and INT (X*10 ↑ D + .5)/10↑D tween two integers up to the larger of the integers).

INT can also be used to round to any specific number of decimal places. For example, INT (10*X + .5)/10↑2 will round X correct to two decimal places, and INT (X*10 ↑ D + .5)/10↑D will round X correct to D decimal places.

3.2.2 The RND Function

The function RND produces a random number between 0 and 1. Note that the argument in this function is not used.

If we want the first twenty random numbers, we write the program below and we get twenty six-digit decimals.

```
10 FOR L = 1 TO 20
20 PRINT RND(1),
30 NEXT L
40 END

RUN

 .406533       .88445        .681969       .939462       .253358
 .863799       .880238       .638311       .602898       .990032
 .570427       .897931       .628126       .613262       .303217
 5.00548 E-2   .393226       .680219       .632246       .668218
```

On the other hand, if we want twenty random one-digit integers, we could change line 20 to read:

```
20 PRINT INT(10*RND(0)),
RUN
```

and we would then obtain:

```
 4             8             6             9             2
 8             8             6             6             9
 5             8             6             6             3
 0             3             6             6             6
```

We can vary the type of random numbers we want. For example, if we want 20 random numbers ranging from 1 to 9 inclusive, we could change line 20 as shown

```
20 PRINT INT(9*RND(1)+1);
RUN

4 8 7 9 3 8 8 6 6 9 6 9 6 6 3 1 4 7 6 7
```

or we can obtain random numbers which are integers from 5 to 24 inclusive by changing line 20 as in the following example:

```
20 PRINT INT(20*RND(1)+5);
RUN

13 22 18 23 10 22 22 17 17 24 16 22 17 17
11  6 12 18 17 16
```

In general, if we want our random numbers to be chosen from the A integers of which B is the smallest, we would call for INT (A*RND(1) + B).

3.2.3 The SGN Function

The SGN function assigns the value 1 to any positive number, 0 to zero, and -1 to any negative number, thus, SGN (7.23) = 1, SGN (0) = 0, and SGN (-.2387) = -1.

3.2.4 DEF Statement

In addition to the standard functions, you can define any other function which you expect to use a number of times in your program by use of a DEF statement. The name of the defined function must be three letters, the first two of which must be FN. Hence, you may define up to 26 functions, e.g., FNA, FNB, etc. The use of a DEF statement is shown in the following example.

```
10 DEF FNX(X) = X * X
20 READ A, B
30 PRINT A, B, FNX(B)
40 GO TO 20
50 DATA 1, 2, 3, 4, 5, 6
60 END

RUN

1             2             4
3             4             16
5             6             36
```

```
1030 ERROR IN LINE  20
OUT OF DATA
READY
```

3.3 GOSUB and RETURN

When a particular part of a program is to be performed more than one time, or possibly at several different places in the overall program, it is most efficiently programmed as a subroutine. The subroutine is entered with a GOSUB statement, where the number is the line number of the first statement in the subroutine. For example,

```
90 GOSUB 210
```

directs BASIC to jump to line 210, the first line of the subroutine. The last line of the subroutine should be a RETURN command directing BASIC to return to the earlier part of the program. For example,

```
350 RETURN
```

will tell BASIC to go back to the first line numbered greater than 90, and to continue the program there.

The following example, a program for determining the greatest common divisor of three integers using the Euclidean Algorithm, illustrates the use of a subroutine. The first two numbers are selected in lines 30 and 40 and their greatest common divisor (GCD) is determined in the subroutine, lines 200-310. The GCD just found is called X in line 60, the third number is called Y in line 70, and the subroutine is entered from line 80 to find the GCD of these two numbers. This number is, of course, the greatest common divisor of the three given numbers and is printed out with them in line 90.

You may use a GOSUB inside a subroutine to perform yet another subroutine. This would be called nested GOSUBs. In any case, it is absolutely necessary that a subroutine be left only with a RETURN statement, using a GOTO or an IF-THEN to get out of a subroutine will not work properly. You may have several RETURNs in the subroutine so long as exactly one of them will be used.

```
 10   PRINT " A", " B", " C", "GCD"
 20   READ A,B,C
 30   LET X=A
 40   LET Y=B
 50   GOSUB 200
 60   LET X=G
 70   LET Y=C
 80   GOSUB 200
 90   PRINT A,B,C,G
100   GO TO 20
110   DATA 60,90,120
```

```
120   DATA 38456,64872,98765
130   DATA 32,384,72
200   LET Q=INT(X/Y)
210   LET R=X-Q*Y
220   IF R=0 GOTO 300
230   LET X=Y
240   LET Y=R
250   GO TO 200
300   LET G=Y
310   RETURN
320   END


RUN

 A             B             C             GCD
 60            90            120            30
 38456         64872         98765          1
 32            384           72             8

1030 ERROR IN LINE   20
OUT OF DATA
READY
```

3.4 INPUT

There are times when it is desirable to have data entered during the running of a program. This is particularly true when one person writes the program and enters it into the computer's memory, and other persons are to supply the data. This may be done by an INPUT statement, which acts as a READ statement but does not draw numbers from a DATA statement. If, for example, you want the user to supply values for X and Y into a program, you will type

```
40 INPUT X, Y
```

before the first statement which is to use either of these numbers. When it encounters this statement, BASIC will type a question mark. The user types two numbers, separated by a comma or blank, presses the RETURN key, and BASIC goes on with the rest of the program.

Frequently an INPUT statement is combined with a PRINT statement to make sure that the user knows what the question mark is asking for. You might type:

```
20 PRINT "YOUR VALUES OF X, Y, AND Z ARE";
30 INPUT X, Y, Z
```

and BASIC will type out

```
YOUR VALUES OF X, Y, AND Z ARE?
```

Without the semicolon at the end of line 20, the question mark would have been printed on the next line.

Data entered via an INPUT statement is not saved with the program. Furthermore, it may take a long time to enter a large amount of data using INPUT. Therefore, INPUT should be used only when small amounts of data are to be entered, or when it is necessary to enter data during the running of the program such as with game-playing programs.

3.5 MISCELLANEOUS STATEMENTS

Several other BASIC statements that may be useful from time to time are STOP, REM and RESTORE.

3.5.1 STOP Statement

STOP is equivalent to GOTO xxxxx, where xxxxx is the line number of the END statement in the program. It is useful in programs having more than one natural finishing point. For example, the following two program portions are equivalent.

```
250 GO TO 999          250 STOP
340 GO TO 999          340 STOP
999 END                999 END
```

3.5.2 REM Statement

REM provides a means for inserting explanatory remarks in a program. BASIC completely ignores the remainder of that line, allowing the programmer to follow the REM with directions for using the program, with identifications of the parts of a long program, or with anything else that he wants. Although what follows REM is ignored, its line number may be used in a GOTO or IF-THEN statement.

```
100 REM INSERT DATA IN LINES 900-998. THE FIRST
110 REM NUMBER IS N, THE NUMBER OF POINTS. THEN
120 REM THE DATA POINTS THEMSELVES ARE ENTERED, BY

200 REM THIS IS A SUBROUTINE FOR SOLVING EQUATIONS
  .
  .
300 RETURN
  .
  .
520 GOSUB 200
```

3.5.3 RESTORE Statement

Sometimes it is necessary to use the data in a program more than once. The RESTORE statement permits reading the data as many additional times as it is used. Whenever RESTORE is encountered in a program, BASIC restores the data to its original state. A subsequent READ statement will then start reading the data all over again. A word or warning: if the desired data are preceded by code numbers or parameters, superfluous READ statements should be used to pass over these numbers. As an example, the following program portion reads the data, restores the data to its original state, and reads the data again. Note the use of line 570 to "pass over" the value of N, which is already known.

```
100 READ N
110 FOR I = 1 TO N
120 READ X
 .
 .
 .
200 NEXT I
 .
 .
 .
560 RESTORE
570 READ X
580 FOR I = 1 TO N
 .
 .
 .
```

CHAPTER 4
OPERATING TSS/8 BASIC SYSTEM

After logging into TSS/8, and calling the BASIC program (see page 1-1), BASIC then types NEW OR OLD? and you type the appropriate adjective: NEW if you are about to type a new problem and OLD if you want to recover a problem on which you have been working earlier and have stored in the computer's memory.

BASIC then asks NEW PROGRAM NAME (or OLD PROGRAM NAME, as the case may be) and you type any combination of letters and digits you like, but no more than six. If you are recalling an old problem from the computer's memory, you must use exactly the same name as that which you gave the problem before you asked BASIC to save it.

BASIC then types READY (which signals the start of the editing phase), and you should begin to type your program. If you type a line consisting of only a line number followed by the RETURN key, that line will be deleted. Make sure that each line begins with a line number which is greater than 0 and less than 2046 and contains no non-digit characters. Also be sure to type the RETURN key at the completion of each line.

If, in the process of typing a statement, you make a typing error and notice it immediately, you can correct it by typing the RUBOUT key (right-hand side of the keyboard). This will delete the character in the preceding space and print a left arrow (←) for each rubout. You can then type in the correct character. Typing the RUBOUT key a number of times will erase from the current line one character (including spaces) to the left for each RUBOUT typed. (Caution: It sometimes takes several seconds for BASIC to accomplish a rubout.) Typing the key marked ALT MODE (left-hand side of the keyboard) will delete the entire line being typed.

While in the editing phase, certain additional commands (which may not have line numbers) are available and are described below:

a. If you type in SAVE followed by typing the RETURN key, the program you have just written (or changed) will be saved for use at a later time, under the name you gave when you started. If following the word SAVE you have typed one or more spaces, followed by a name followed by the RETURN key, the program will be saved under that name.

b. If you type in UNSAVE, followed by a name, followed by the RETURN key, the program with the name you have just given will be deleted from your permanent file.

c. If you type in CATALOG followed by the RETURN key, a listing of all the program names in your permanent file will be typed.

NOTE

Names of temporary files may also be shown. These will be of the form, BAS_n, and may be ignored.

d. If you type in LIST followed by two line numbers separated by a comma, a listing of that part of your current program, which lies between those line numbers, will be typed. If the comma and second line number are omitted, only the single line indicated will be listed. If no line numbers follow the word LIST (but only the RETURN key), your whole program will be listed.

e. If you type in DELETE followed by two line numbers separated by a comma, all lines, between the two indicated, will be deleted. If the comma and second line number are omitted, only the single line specified will be omitted.

f. When you are ready to leave the Teletype, sign off by typing BYE. This concludes operations and TSS/8 deletes any temporary files assigned to you.

The editor phase control words (CATALOG, DELETE, LIST, NEW, OLD, RUN, SAVE, UNSAVE and BYE) may either be typed as shown or the equivalent effect may be obtained by typing in just the first letter. If information (other than the RETURN key) is to follow the control word, at least one blank must follow the control word (or character).

After typing your complete program, you type RUN, press the RETURN key; BASIC will then analyze your program. If the program is one which BASIC can run, it will type out any results for which you have asked in your PRINT statements. This does not mean that your program is correct, but that it has no errors of the type known as syntactical or format errors. If it has errors of this type, BASIC will type an error message to you and return to the editing phase. A list of the error messages with the interpretation of each is contained in Appendix C.

If it is obvious that you are getting the wrong answers to your problem, even while the program is running, you can press the ALT MODE key and computation will cease. The computer will than type READY to indicate that you are back in the editing phase. It may be necessary to type CTRL/B followed by S;ST 222↵ in order to stop processing and return to the Editor.

APPENDIX A

SUMMARY OF BASIC STATEMENTS

A.1 ELEMENTARY BASIC STATEMENTS

The following subset of the Dartmouth BASIC command repertoire includes the most commonly used commands and is sufficient for solving most problems.

LET [variable] = [formula]	Assign the value of the formula to the specified variable.
DATA [data list]	DATA statements are used to supply one or more numbers to be accessed by READ statements.
READ [variable list]	READ statements, in turn, assign the next available datum in the DATA string to the variables listed.
PRINT [arguments]	Type the values of the specified arguments, which may be variables, text, or format control characters.
GO TO [line number]	Transfer control to the line number specified and continue execution from that point.
IF [formula] [relation] [formula] $\left\{\begin{matrix}\text{THEN}\\ \text{GO TO}\end{matrix}\right\}$ [line number]	If the stated relationship is true, then transfer control to the line number specified; if not, continue in sequence.
FOR [variable] = [formula$_1$] TO [formula$_2$] STEP [formula$_3$]	Used for looping repetitively through a series of steps. The FOR statement initializes the variable to the value of formula$_1$. If the increment is positive and the variable $\leq$ formula$_2$, the instructions following are executed until the NEXT statement is encountered.
NEXT [variable]	The NEXT statement increments the variable by the value of formula$_3$ (if omitted, the increment value is +1). The variable is again tested as described above, and this process continues until the loop is repeated the specified number of times. When the variable becomes larger than formula$_2$, control goes to the statement following the NEXT. If the increment (formula$_3$) is negative, then the instructions between the FOR and NEXT statements are executed until the variable becomes less than the value of formula$_2$.
DIM [variable] [subscript]	Enables the user to enter a table or array with the specified number of elements.
END	Last statement to be executed in the program. This statement must be present.

FORMULAS: In addition to the common arithmetic operators of addition (+), subtraction (-), multiplication (*), division (/), and exponentiation (↑), BASIC includes the following elementary functions:

SIN (x)		LOG (x)
COS (x)	ATN (x)	ABS (x)
TAN (x)	EXP (x)	SQR (x)

A.2 ADVANCED BASIC STATEMENTS

GOSUB [line number] Subroutine { [line number] ... RETURN	Simplifies the execution of a subroutine at several different points in the program by providing an automatic return from the subroutine to the next sequential statement following the appropriate GOSUB (the GOSUB which sent control to the subroutine).
INPUT [variable(s)]	Causes typeout of a ? to the user and waits for user to respond by typing the value(s) of the variable(s).
STOP	Equivalent to GO TO [line number of END statement].
REM	Permits typing of remarks within the program.
RESTORE	Sets pointer back to beginning of string of DATA values.

FORMULAS: Some advanced functions include the following:

INT (x)	Find the greatest integer in x.
RND (x)	Generate random numbers between 0 and 1. The same set of random numbers can be generated repeatedly for purposes of program testing and debugging. The value of x is ignored.
SGN (x)	Assign a value of 1 if x is positive; 0 if x is 0; or -1 if x is negative.

The user can also define his own functions by use of the DEFine statement. For example,

[line number] DEF FNC(x) = SIN(x) + TAN(x) -10

(Defines the user function FNC as the formula SIN(x) + TAN(x) -10.)

NOTE that DEFine statements are restricted to one line.

APPENDIX B

SUMMARY OF EDIT AND CONTROL COMMANDS

Several commands for editing BASIC programs and for controlling their execution enable you to: delete lines, list your program, save programs on a file-structured storage device (disk), delete or replace old programs on the storage device with new programs, call in programs from the storage device, etc. These commands are summarized below.

Command	Action
BYE	Exit to TSS/8 Monitor to conclude operations.
DELETE n (or n↵)	Delete line number n (or simply the line number and RETURN key).
DELETE n_1, n_2	Delete line numbers n_1 through n_2.
LIST	List program
LIST n	List line number n.
LIST n_1, n_2	List program from line number n_1 through n_2.
NEW	BASIC will ask for new program name.
OLD	BASIC will ask for program name and will replace current contents of user core with existing program of that name from the storage device.
RUN	Compile and run program currently in core.
SAVE	Save the contents of user core as file whose filename is current program name.
SAVE [name]	Save user core as name[1].
UNSAVE	Delete the program with the current program name from the storage device.
UNSAVE [name]	Delete the name program from the storage device.
↑B (CTRL/B)	To stop a running program, type ↑B followed by S; ST 222↵. CTRL/B is typed by holding down the CTRL key while typing the B key; it echoes the ↑B on the Teletype printer.

[1]SAVE commands will not overwrite an existing file of the same name (use UNSAVE first).

APPENDIX C
ERROR MESSAGES

Four types of error messages can occur in BASIC. The messages and their interpretation is shown below.

C.1 DURING THE EDITING PHASE

(Just retype the line to correct it.)

//ERROR 00	Machine malfunction.
//ERROR 01	You didn't type in OLD or NEW when the information was requested.
//ERROR 02	The new or old name you typed in wasn't a valid name.
//ERROR 03	The new name you gave is already an active program.
//ERROR 04	You asked for an old program name which isn't in your permanent file.
//ERROR 10	The name you gave with the SAVE command is already in your permanent file.
//ERROR 11	The SAVE or UNSAVE name you gave is not a valid name.
//ERROR 12	The editor can't understand the command you just gave.
//ERROR 20	Invalid line number format or outside of the range 0 < line number < 2047.

C.2 DURING PROGRAM COMPILATION OR EXECUTION

The message will be preceded by ERROR ON LINE nnnn, where nnnn is the line number on which the error was detected. (BASIC will type READY and you will be back in the editing phase.)

PROGRAM TOO LARGE TO LOAD	Your program is too large to be executed. Try to make it smaller.
MISSING END STATEMENT	All programs must have an END statement.
DATA POOL OVERFLOW	You have used too many constants and/or variables in your program.
ILLEGAL STATEMENT	A statement was used which is not one of the legitimate BASIC statements.
ILLEGAL LINE FORMAT	The structure of the statement does not agree with the BASIC syntax.
NOT CONSTANT IN DATA	You attempted to use something other than a constant in a DATA statement.

ILLEGAL CHARACTER	You attempted to use an illegal character for the statement you are processing.
ILLEGAL CONSTANT	The format of a constant, in the statement being processed, is not valid.
INVALID NAME	A name is being used which doesn't agree with the BASIC requirements.
INVALID LINE NUMBER	The format of the line number, being used in a GO TO or IF statement, is not correct.
ARRAY USED BEFORE DEFINED	You have attempted to use an array prior to its appearance in a DIM statement.
EXPRESSION SYNTAX	The expression being processed does not agree with the BASIC rules (probably this will be due to unmatched parentheses).
STACK OVERFLOW	You have programmed a situation in which either DO, subroutines, or functions are nested too deeply or you have a function or subroutine which calls itself.
OUT OF DATA	An attempt has been made to READ more data than you have.
ILLEGAL INPUT FORMAT	The form of a constant, which you are attempting to INPUT, is not valid.
DIMENSION SIZE	Too large an array.
UNDEFINED LINE NUMBER	The line number appearing in a GO TO or an IF-THEN statement does not appear in the program.

C.3 NON-FATAL EXECUTION ERRORS

These errors are for notification purposes and indicate that you have performed a computational range error. They will all type the message XX IN nnnn, where nnnn is the line number and XX is as described below.

Error Code	Explanation
/0	ZERO DIVIDE - An attempt was made to divide a number by zero. The largest possible number will be used for the result.
OV	OVERFLOW - The result of a calculation was too large for the computer to handle. The largest possible number will be used for the result.
UN	UNDERFLOW - The result of a calculation was too small for the computer to handle. Zero will be used for the result.
LN	An attempt was made to compute the logarithm of zero or a negative number. Zero will be used for the result.

SQ	An attempt was made to compute the square root of a negative number. The square root of the absolute value will be used for the result.
PW	An attempt was made to raise a negative number to a power. The absolute value raised to a power will be used.

C.4 SYSTEM ERROR

If a failure occurs in the I/O portion of the BASIC system, the message MACHINE MALFUNCTION will be typed and control will return to the editing phase.

APPENDIX D

D.1 LOADING TO DISK

The system call routine (SYP) references a table called SYPTBA at EXEC locations $314\text{-}332_8$. This table contains the disk address of the five (5) basic programs. As is shown in the table, the programs are on the disk in the following order:

EDITOR	(BASED)
COMPILER	(BASCOM)
LOADER	(BASLDR)
INTERPRETER	(BASIN)
ERROR	(BASERR)

Having gone once through all the preliminaries of Opening, Creating, Extending and setting the Protection Mask for BASIC, you can start with the loading of the disk.

Load the EDITOR/EXEC (binary tape) using R LOAD.

Check the AC on completion with a WHERE. If the AC $\neq 0$ reload the tape.

If AC=0 perform a SAVE BASIC 0 0 4377. EDITOR will now occupy locations 0-4377 on disk. The compiler is loaded next. Perform WHERE. If AC=0 perform a

SAVE BASIC 4400 — First disk location used by the COMPILER

400 5377 — Starting location and 2's complement of the length of the COMPILER

This will butt the COMPILER up against the EDITOR/EXEC on the disk. This same procedure is used for each of the remaining programs. The succeeding SAVE's are:

LOADER:	SAVE	BASIC	11400	400	777
INTERPRETER:	SAVE	BASIC	12000	400	6477
ERROR:	SAVE	BASIC	20100	400	1777

NOTE

Changes in the size of these programs will require alterations to the table entries in SYPTBA and in the SAVE's performed in loading the disk.

APPENDIX E

IMPLEMENTATION NOTES

TSS/8 BASIC language is compatible with Dartmouth BASIC except as noted below:

1. There are no matrix operations.

2. There are no character string instructions.

3. The ON statement has not been implemented.

The TAB function is not available in PRINT statements.

5. BASIC has no features which allow reading or writing data on the disk. (Although programs may be saved on the disk for future use.)

6. All array (subscripted) variables must appear in a DIM statement.

7. The function INT(x) will give the greatest integer in x. Thus INT(-2.3) will give the value -2.

8. Negative numbers may not be raised to integer powers. The absolute value will be used and an error message will be printed.

9. The RANDOMIZE instruction is not available.

10. User defined functions are restricted to one line.

HOW TO OBTAIN REVISIONS AND CORRECTIONS

Notification of new or revised DEC software and manuals available from the Program Library is published in:

Digital Software News for the PDP-8 Family
Digital Software News for the PDP-9 Family

If you are not receiving the publication appropriate to your computer, please notify Software Information Service (see Reader's Comments card).

Revised software products and documents are shipped only after the Program Library receives a specific request from a user (see title page for address).

Digital Equipment Computer User's Society (DECUS) maintains a library of user software and publishes them in DECUSCOPE, a magazine available to both DECUS members and to non-members who request it. Return the request card below to receive further information or to place your name on the mailing list.

To: Decus Office,
Digital Equipment Corporation,
Maynard, Massachusetts 01754

☐ Please send DECUS installation membership information.

☐ Please send DECUS individual membership information.

☐ Please add my name to the DECUSCOPE non-member mailing list.

Name _____

Company _____

Address _____

_____

(Zip Code)

Digital Equipment Corporation
Maynard, Massachusetts

digital

printed in U.S.A.