Digital Equipment Corporation
Maynard, Massachusetts

digital

PDP-10

ADVANCED BASIC

DEC-10-KJZA-D

ADVANCED **BASIC** FOR THE **PDP-10**

For additional copies order No. DEC-10-KJZA-D from Program Library, Digital Equipment Corporation, Maynard, Mass. Price $3.50

DIGITAL EQUIPMENT CORPORATION □ MAYNARD, MASSACHUSETTS

Printed in U.S.A.

1st Edition November 1968

Copyright © 1968 by Digital Equipment Corporation

BASIC® was developed by Dartmouth College, Hanover, New Hampshire and is copyrighted by the Trustees of Dartmouth College. We would like to thank Dartmouth College for the privilege of using their BASIC Manual, 4th Edition as a format for this manual.

Instruction times, operating speeds and the like are included in this manual for reference only; they are not to be taken as specifications.

The following are registered trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

DEC	PDP
FLIP CHIP	FOCAL
DIGITAL	COMPUTER LAB

® Registered: Trustees of Dartmouth College

FOREWORD

BASIC[R] is a conversational, problem-solving language for scientific, business and educational applications. It is used to solve simple and complex mathematical problems from your Teletype console, and it is particularly suited to time-sharing.

With BASIC, you type your computational procedure as a series of numbered statements, utilizing common English syntax and familiar mathematical notation. If BASIC is new to you, you need spend only an hour or so learning the elementary commands necessary for solving almost any problem. With experience, you may add the advanced commands needed to perform more intricate manipulations and to express your problems more efficiently and concisely.

Once you have entered your statements via your console, simply type RUN to initiate execution of your routine and receive your results instantaneously.

[R] Registered: Trustees of Dartmouth College

CONTENTS

Page

CHAPTER 1
INTRODUCTION

1.1	Special Features	1-1
1.2	Example of a BASIC Program	1-1
1.3	Formulas	1-6
1.3.1	Arithmetic Operations	1-6
1.3.2	Mathematical Functions	1-7
1.3.3	Numbers	1-7
1.3.4	Variables	1-8
1.3.5	Relational Symbols	1-8
1.4	Loops	1-8
1.4.1	FOR and NEXT Statements	1-9
1.4.2	Nested Loops	1-11
1.5	Lists and Tables	1-11
1.5.1	DIM (Dimension) Statement	1-11
1.5.2	Examples	1-12
1.6	Running BASIC	1-14
1.6.1	Gaining Access to BASIC	1-14
1.6.2	Entering Your Program Statements	1-15
1.6.3	Executing Your Program	1-16
1.6.4	Correcting Your Program	1-15
1.6.5	Interrupting the Execution of Your Program	1-16
1.6.6	Leaving the Computer	1-16
1.6.7	Example of BASIC Run	1-17
1.7	Errors and Debugging	1-18
1.7.1	Example of Finding and Correcting Program Errors	1-19
1.8	Summary of Elementary Basic Statements	1-21
1.8.1	LET Statement	1-22
1.8.2	READ and DATA Statements	1-22
1.8.3	PRINT Statement	1-23
1.8.4	GO TO Statement	1-24

CONTENTS (Cont)

		Page
1.8.5	IF -- THEN Statement	1-24
1.8.6	ON...GO TO Statement	1-24
1.8.7	FOR and NEXT Statements	1-25
1.8.8	DIM Statement	1-26
1.8.9	END Statement	1-26

CHAPTER 2
ADVANCED BASIC STATEMENTS

2.1	More About The Print Statement	2-1
2.2	INT, RND, and SGN Functions, and the DEF Statement	2-4
2.2.1	The INT (Integer) Function	2-4
2.2.2	The RND (Random Number Generating) Function	2-4
2.2.3	The RANDOMIZE Statement	2-6
2.2.4	The SGN (Sign) Function	2-6
2.2.5	The DEF (Define User Function) and FNEND (Function End) Statements	2-7
2.3	Subroutines: GOSUB and RETURN Statements	2-8
2.4	INPUT Statement	2-9
2.5	STOP, REM (Remarks), and RESTORE Statements	2-10
2.5.1	STOP Statement	2-10
2.5.2	REM (Remarks) Statement	2-10
2.5.3	RESTORE Statement	2-11
2.6	Matrices	2-11
2.6.1	MAT Statement Conventions	2-12
2.6.2	MAT c = ZER, MAT c = CON, MAT c = IDN	2-13
2.6.3	MAT PRINT a, b, c	2-14
2.6.4	MAT INPUT v and the NUM Function	2-15
2.6.5	MAT b = a	2-15
2.6.6	MAT c = a + b and MAT c = a -b	2-16
2.6.7	MAT c = a * b	2-16
2.6.8	MAT c = TRN (a)	2-16
2.6.9	MAT c = (k) * a	2-16
2.6.10	MAT c = INV(a) and the DET Function	2-16

CONTENTS (Cont)

		Page
2.6.11	Examples of Matrix Programs	2-17
2.6.12	Simulation of n-Dimensional Arrays	2-18
2.7	Alphanumeric Information (Strings)	2-19
2.7.1	Reading and Printing Strings	2-20
2.7.2	String Conventions	2-21
2.7.3	Numeric and String DATA Blocks; RESTORE* RESTORE$ Statements	2-22
2.7.4	Accessing Individual Characters; the CHANGE Statement	2-22
2.8	Diagnostic Messages	2-24
2.9	Edit and Control Commands	2-25

APPENDIX A
SUMMARY OF BASIC STATEMENTS

CHAPTER 1

INTRODUCTION

1.1 SPECIAL FEATURES

Advanced BASIC incorporates the following special features:

a. Matrix Computations - A special set of 13 commands designed for performing matrix computations are included.

b. Alphanumeric Information Handling - Single and/or vectors of alphabetic/alphanumeric strings can be read, printed, and defined in LET and IF...THEN statements. In addition, individual characters within these strings can be easily accessed by the user. Conversion can be performed between characters and their ASCII equivalents. Also, tests can be made for alphabetic order.

c. Program Control and Storage Facilities - Facilities store programs or data on a mass storage device (e.g., disk or DECtape) and later retrieve them for execution. The user can also input his program from the standard, low-speed Teletype paper tape reader as well as the high-speed paper tape reader at the PDP-10 site.

d. Program Editing Facilities - An existing program can be edited by adding or deleting lines, renaming the program, or resequencing the line numbers. The user can merge two programs into a single program and request a listing of his program, either in whole or in part, on his Teletype or a high-speed line printer.

e. Formatting of Output - Controlled formatting of Teletype output includes tabbing, spacing, and the printing of columnar headings.

f. Documentation and Debugging Aids - The insertion of remarks enables recall of needed information at some later date. Debugging of programs is aided by the typeout of meaningful diagnostic messages pinpointing syntactical and logical errors detected during execution.

1.2 EXAMPLE OF A BASIC PROGRAM

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

$$ax + by = c$$

$$dx + ey = f$$

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

If ae - bd is not equal to 0, this system can be solved to find that

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

If ae - bd = 0, there is either no solution or many, but there is no unique solution. Study this example carefully -- in most cases the purpose of each line in the program is self-evident -- and 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↲
```

NOTE

All statements are terminated by pressing the RETURN key (represented in this text by the symbol ↲). The RETURN key echoes as a carriage return, line feed.

Each line of the program begins with a line number and serves to identify each line as a statement. A program is made up of such statements, most of which are instructions to the computer. Line numbers serve to specify the order in which these statements are to be performed. Before the program is run, BASIC sorts out and edits the program, putting the statements into the order specified by their line numbers. This means that the program statements can be typed in any order, as long as each statement is prefixed with a line number indicating its proper sequence in the order of execution. Each statement starts after its line number with an English word which denotes the type of statement. Spaces have no significance in BASIC, except in messages which are printed out, as in line number 65 above. Thus, spaces may be used, or unused, at will to modify a program and make it more readable.

With this preface, the above example can be followed through step-by-step.

```
10 READ A, B, C, D
```

The first statement, 10, is a READ statement and must be accompanied by one or more DATA statements. When the computer encounters a READ statement while executing a program, it will cause the variables listed after the READ to be given values according to the next available numbers in the DATA statements. In this example, we read A in statement 10 and assign the value 1 to it from statement 70 and,

similarly, with B and 2, and with D and 4. At this point, the available data in statement 70 has been exhausted, but there is more in statement 80, and we pick up from it the value 2 to be assigned to E.

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

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

```
20 IF G = 0 THEN 65
```

If G is equal to zero, the system has no unique solution. Therefore, we next ask, in line 20, if G is equal to zero.

```
65    PRINT "NO UNIQUE SOLUTION"
70    DATA 1, 2, 4
80    DATA 2, -7, 5
85    DATA 1, 3, 4, -7
90    END
```

If the computer discovers a "yes" answer to the question, it is directed to go to line 65, where it prints "NO UNIQUE SOLUTION." Since DATA statements are not "executed", it then goes to line 90 which tells it to "END" the program.

```
30 READ C, F
```

If the answer to the question "Is G equal to zero?" is "no", the computer goes to line 30. The computer is now directed to read the next two entries, -7 and 5, from the DATA statements (both are in statement 80) and to assign them to C and F respectively. The computer is now ready to solve the system.

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

```
37 LET X = (C*E - B*F) / G
42 LET Y + (A*F - C*D) / G
```

In statements 37 and 42, we compute the value of X and Y according to the formulas provided using parentheses to indicate that CE - BF is divided by G.

```
55 PRINT X, Y
60 GO TO 30
```

The computer prints the two values X and Y, in line 55. Having done this, it moves on to line 60 where it is reverted to line 30. With additional numbers in the DATA statements, the computer is told in line 30 take the next one and assign it to C, and the one after that to F. Thus,

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

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

In line 30 the computer reads two more values, 4 and -7, which it finds in line 85. It then proceeds to solve the system

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

and print out the solutions. Since there are no more pairs of numbers in the DATA statement available for C and F, the computer prints "OUT OF DATA IN 30" and stops.

If we had omitted line number 55 (PRINT X, Y) the computer would have solved the three systems and then told us when it was out of data. Had we omitted line 20 and G were equal to zero, the computer would print "DIVISION BY ZERO IN 37" and "DIVISION BY ZERO IN 42." Had we left out statement 60 (GO TO 30), the computer 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."

The particular choice of line numbers is arbitrary as long as the statements are numbered in the order the machine is to follow. We would normally number the statements 10, 20, 30, ..., 130, so that we can later insert additional statements. Thus, if we find that we have left out two statements between those numbered 40 and 50, we can give them any two numbers between 40 and 50 -- say 44 and 46. In regards to DATA statements, 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.

```
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 ↲

LINEAR              11:03              10/19/68

4                   -5.50000
0.666667             0.166667
-3.66667             3.83333
OUT OF DATA IN 30
```

NOTE

Typeouts from BASIC or from the Monitor are indicated in this text by underscoring.

After typing the program, we type RUN followed by a carriage return which directs the computer to execute the program. Note that the computer, before printing out the answers, printed the name "LINEAR" which we gave to the problem (see Section 1.6) and the time and date of the computation. The message "OUT OF DATA IN 30" here may be ignored. However, in some cases it indicates an error in the program (see Section 1.7.2).

1.3 FORMULAS

The computer can perform innumerable operations: add, subtract, multiply, divide, extract square roots, raise a number to a power, find the sine of a number (or an angle measured in radians), etc.

1.3.1 Arithmetic Operations

The computer performs its primary function (that of computation) by evaluating formulas similar to those used in standard mathematical calculation, with the exception that all BASIC formulas must be written on a single line. Five arithmetic operations can be used to write a formula.

Symbol	Example	Meaning
+	A + B	add B to A
-	A - B	subtract B from A
*	A * B	multiply B by A
/	A / B	divide A by B
↑	X ↑ 2	find X^2

If we type A + B * C ↑ D, the computer will first raise C to the power D, multiply this result by B, and then add A to the resulting product. We must use parentheses to indicate any other 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 add A to B, multiply their 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:

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

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

c. In the absence of parentheses in a formula involving only multiplication and division, the operations are performed from left to right, in the order that they are read. So also does the computer perform addition and subtraction from left to right.

The rules tell us that the computer, faced with A - B - C, will (as usual) subtract B from A and then C from their difference; faced with A/B/C, it will divide A by B and that quotient by C. Given A ↑ B ↑ C, the computer will raise the number A to the power B and take the resulting number and raise it to the power C. If there is any question in your mind about the priority, put in more parentheses to eliminate possible ambiguities.

1.3.2 Mathematical Functions

In addition to these five arithmetic operations, the computer can evaluate several mathematical functions. These functions are given special 3-letter English names.

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	
COT (X)	Find the cotangent of X	
ATN (X)	Find the arctangent of X	
EXP (X)	Find e^X	
LOG (X)	Find the natural logarithm of X (ln X)	
ABS (X)	Find the absolute value of X ($\|X\|$)	
SQR (X)	Find the square root of X ($\sqrt{X}$)	

Five other functions are also available in BASIC: INT, RND, SGN, NUM, and DET; these are reserved for explanation in Chapter 2. In place of X, we may substitute any formula or any number in parentheses following any of these formulas. For example, we may ask the computer to find $\sqrt{4+X^3}$ by writing SQR (4 + X ↑ 3), or the arctangent of $3X - 2e^X + 8$ by writing ATN (3 * X - 2 * EXP (X) + 8).

If the value of $(\frac{5}{6})$ 17 is needed, the 2-line program can be written

```
10    PRINT (5/6) ↑17
20    END
```

and the computer will find the decimal form of this number and print it out.

1.3.3 Numbers

A number may be positive or negative and it may contain up to eight digits, but it must be expressed in decimal form (i.e., 2, -3.675, 12345678, -.98765432, and 483.4156). The following are not numbers in BASIC: 14/3, $\sqrt{7}$, and .00123456789. The computer can find the decimal expansion of 14/3 or $\sqrt{7}$, but we may not include either in a list of DATA. We gain further flexibility by use of the letter E, which stands for "times ten to the power." Thus, we may write .0012345678 as .12345678E-2 or 12345678E-11 or 1234.5678E-6. We do not write E7 as a number, but write 1E7 to indicate that it is 1 that is multiplied by 10^7.

1.3.4 Variables

A numerical variable in BASIC is denoted by any letter, or by any letter followed by a single digit[1]. Thus, the computer will interpret E7 as a variable, along with A, X, N5, I0, and O1. 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 0 before a RUN. Thus, it is only necessary to assign a value to a variable when a value other than 0 is required.

Although the computer does little in the way of "correcting" during computation, it will sometimes help if an absolute value hasn't been indicated. For example, if you ask for the square root of -7 or the logarithm of -5, the computer will give the square root of 7 with the error message that you have asked for the square root of a negative number, or it will give the logarithm of 5 with the error message that you have asked for the logarithm of a negative number.

1.3.5 Relational Symbols

Six other mathematical symbols of relation 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 sample program in Section 1.

Any of the following six standard relations may be used:

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

1.4 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, usually with slight changes each time. In order to write the simplest program in which the portion to be repeated is written just once, we use a loop.

---

[1] In this chapter we will discuss only numerical variables. See Section 2.7 for alphanumeric "string variables."

The programs which use loops can, perhaps, be best illustrated and explained by two versions of a program for the simple task of printing out a table of the positive integers 1 through 100 together with the square root of each. Without a loop, our 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 far fewer lines of instruction, 5 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 the computer back to line 20. Here it 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. The computer now receives a negative answer to the question in line 40 (X is greater than 100, not less than or equal to it), does not return to 20 but moves on to line 50, and ends the program. All loops contain four characteristics: initialization (line 10), the body (line 20), modification (line 30) and an exit test (line 40).

1.4.1 FOR and NEXT Statements

BASIC provides two statements to specify a loop, the FOR and NEXT statements.

```
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 above. Line 30 carries out two tasks: X is increased by 1, and the text is carried out 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.

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

```
10  FOR X = 1 TO 100 STEP 5↵
```

and the computer would assign 1 to X on the first time through the loop, 6 to X on the second time, 11 on the third, and 96 on the last time. Another step of 5 would take X beyond 100, so 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 ↵
```

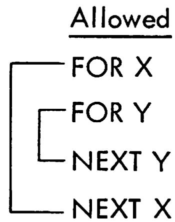
For a positive step-size, the loop continues as long as the control variable is less than or equal to the final value. For a negative step-size, the loop continues as long as the control variable is greater than or equal to the final value.

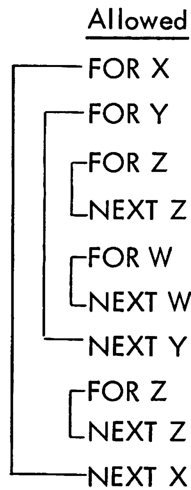
If the initial value is greater than the final value (less than, for negative step-size) the body of the loop will not be performed at all, but the computer will immediately pass to the statement following the NEXT. As an example, the following program for adding up the first n integers will give the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

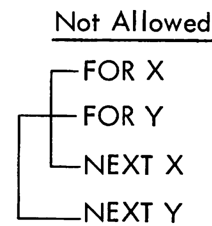
```
10      READ N ↵
20      LET S = 0↵
30      FOR K = 1 TO N↵
40      LET S = S + K ↵
50      NEXT K ↵
60      PRINT S ↵
70      GO TO 10↵
90      DATA 3, 10, 0↵
99      END↵
```

1.4.2 Nested Loops

Nested loops, loops within loops, can be expressed with FOR and NEXT statements. They must be nested and not crossed as the following skeleton examples illustrate:







1.5 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 of a table. Lists are used where we might ordinarily use a single subscript, and tables are used where we might use 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 consist of a single letter, which we call the name of the list or table followed by the subscripts in parentheses. Thus, we might write A(0), A(1), A(2), etc., for the coefficients of the polynomial and B(1,1), B(1,2), etc. for the elements of the matrix.

We can enter the list A(0), A(1),...A(10) into a program very simply by the lines:

```
10  FOR I = 0 TO 10↲
20  READ A(I)↲
30  NEXT I↲
40  DATA 2, 3, -5, 2.2, 4, -9, 123, 4, -4, 3↲
```

1.5.1 DIM (Dimension) Statement

We need no special instruction to the computer if no subscript greater than 10 occurs. However, if we want larger subscripts, we must use a DIM statement, to indicate to the computer that it

has to save extra space for the list or table. When in doubt, indicate a larger dimension than you expect to use. For example, if we want a list of 15 numbers entered, we might write

```
10  DIM A(25)↲
20  READ N↲
30  FOR I = 1 TO N↲
40  READ A(I)↲
50  NEXT I↲
60  DATA 15↲
70  DATA 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47↲
```

Statements 20 and 60 could have been eliminated by writing 30 as FOR I = 1 TO 15, but the program as typed allows for the lengthening of the list by changing only statement 60, so long as it does not exceed 25.

We would enter a 3x5 table into a program by writing

```
10  FOR I = 1 TO 3↲
20  FOR J = 1 TO 5↲
30  READ B (I,J)↲
40  NEXT J↲
50  NEXT I↲
60  DATA 2, 3, -5, -9, 2↲
70  DATA 4, -7, 3, 4, -2↲
80  DATA 3, -3, 5, 7, 8↲
```

Here again, we may enter a table with no DIM statement, and it will handle all the entries from B(0,0) to B(10,10). If you try to enter a table with a subscript greater than 10, without a DIM statement, you will get an error message telling you that you have a subscript error. This is easily rectified by entering the line

```
5 DIM B(20,30)↲
```

if, for instance, we need a 20-by-30 table.

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(I+K) or the table items B(I,K) or Q(A(3,7), B - C).

1.5.2 Examples

Below are the statements and run of a problem which use both a list and a table. The program computes the total sales 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. Product 1 sells for $1.25 per item, product 2 for $4.30 per item, and product 3 for $2.50 per item; also, salesman 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 through 80, using data in lines 910 through 930. The same program could be used again, modifying only line 900 if the prices change, and only lines 910 through 930 to enter the sales in another month.

This sample program did not need a DIM statement, since the computer automatically saves enough space to allow all subscripts to run from 0 to 10. A DIM statement is normally used to save more space, but in a long program, requiring many small tables, DIM may be used to save less space for tables, in order to leave more for the program.

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

```
 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↲
999  END↲
```

NOTE

Statements may be indented for visual identity of the various loops within the program.

```
READY
RUN↲

SALES1          11:06          10/20/68
```

TOTAL SALES FOR SALESMAN 1 $180.500
TOTAL SALES FOR SALESMAN 2 $211.300
TOTAL SALES FOR SALESMAN 3 $131.650
TOTAL SALES FOR SALESMAN 4 $166.550
TOTAL SALES FOR SALESMAN 5 $169.400

1.6 RUNNING BASIC

1.6.1 Gaining Access to BASIC

You can gain access to BASIC by first performing any steps required for obtaining service from the Monitor. In the case of the Time-sharing Monitors (see Section 1.5.7 for an example of running under a time-sharing Monitor), such steps include logging into the system (see the PDP-10 System User's Guide or the Time-Sharing Monitors: Multiprogramming Monitor (10/40), Swapping Monitor (10/50) manual). Then, once the Monitor has responded with a period to indicate it is ready to receive a Monitor command, type

.R BASIC↵ (or RUN dev: BASIC, if the BASIC is not on the system device, SYS:)

BASIC then responds with

NEW OR OLD --

Respond with

NEW↵

if you are about to create a new program; BASIC will then ask you for the name of your new program and will check to see that the name does not already exist. Or, if you want to work with a previously created program, type

OLD↵

BASIC will then ask for the name of the program and will replace the current contents of user core with the program of that name from the storage device (disk or DECtape). Program names can be any combination of letters and digits (characters other than letters or digits may be used, but * , ; / $ should be avoided) but must not exceed six characters in length. In the previous examples in this manual, we have used program names such as LINEAR and SALES1. If you are recalling an old program, you must use exactly the same name as the one you assigned the program when you saved it (see Section 2.9).

1.6.2 Entering Your Program Statements

BASIC then responds with

READY

and you can now begin to type your program. Make sure that each line begins with a line number containing no more than five digits and containing no spaces or non-digit characters. Also be sure to start at the beginning of the Teletype line. Press the RETURN key upon completion of each line.

If, in the process of typing a statement, you make a typing error and notice it before you terminate the line, you can correct it by pressing the RUBOUT key once for each character position to be erased, going backwards until you reach the character in error. Then continue typing, beginning with the character in error.

```
10 PRNIT \\\INT 2, 3↲        (Note that the RUBOUT key echoes
                              as a backslash \)
```

Also, you can press the ALTMODE key (if a Teletype Model 35) or the ESC key (if a Teletype Model 33) or the PREFIX key (if a Teletype Model 37) to delete the entire line being typed.

1.6.3 Executing Your Program

After typing your complete program (do not forget to end with an END statement), type RUN followed by the RETURN key. BASIC types the name of your program, the time of day, the current date, and then analyzes your program. If your program is runnable, BASIC will execute it and type out any results you requested via PRINT statements. The typeout of results does not guarantee that your program is correct (the results could be wrong), but it does mean that there were no "grammatical" errors (e.g., missing line numbers, misspelled words, illegal syntax). If there are errors of this type, BASIC types a message (or several messages) to you. A list of these diagnostic messages is given in Section 2.8 with their meanings.

1.6.4 Correcting Your Program

If you receive an error message typeout informing you, for example, that line 60 is in error, you can correct it by typing in a new line 60 to replace the erroneous one. If you want to eliminate the statement on line 110 from your program, you can do this by typing

```
110
```

followed by the RETURN key. If you want to insert a statement between lines 60 and 70, you can do this by typing a line number between 60 and 70 (e.g., 65) followed by the statement.

1.6.5 Interrupting the Execution of Your Program

If, for some reason, the results being typed out seem to be incorrect, and you want to stop the execution of your program, you type

↑C (Hold down the CTRL key and at the same time depress the C key.)

Monitor responds with a period and waits for you to type a Monitor command. If you wish to reinitialize, type either

. START ↵ or . REENTER ↵

BASIC will respond with

READY

whereupon you can modify or add statements and/or type RUN. If you wish to continue at the point where you interrupted the execution, type

. CONT ↵

If you want to run some program other than BASIC, type

. R progname ↵

if the program is on the system device (SYS:), or

. RUN dev: progname ↵

if the program is on some other device.

1.6.6 Leaving the Computer

When you wish to leave the computer, type

↑C

Monitor responds with a period. Now type

. KJOB ↵

Monitor responds with

CONFIRM:

If you simply want to get off the machine and delete all files you may have created, type

K↵

Other options available following the typeout of CONFIRM: are listed for you if you respond to the CONFIRM: message with a carriage return (RETURN key) only. The Monitor will then list all options available along with the response required to request each option.

1.6.7 Example of BASIC Run

A simple example of using BASIC under a time-sharing Monitor is given below:

.↑C	GO TO MONITOR LEVEL
.LOGIN ↵	REQUEST LOGIN
JOB 7 3.19 U	MONITOR TYPES OUT YOUR ASSIGNED JOB NUMBER, THE CURRENT VERSION NUMBER OF THE MONITOR
#27,20 ↵	MONITOR REQUESTS YOUR PROJECT-PROGRAMMER NUMBER; TYPE IT IN
XXXX ↵	MONITOR TYPES OUT MASK; TYPE YOUR PASSWORD OVER IT
0927 29-OCT-68 TTY3	MONITOR TYPES OUT THE TIME OF DAY, THE CURRENT DATE, AND YOUR TELETYPE UNIT NUMBER
.R BASIC ↵	INSTRUCT MONITOR TO BRING BASIC INTO CORE AND START ITS EXECUTION
NEW OR OLD -- NEW↵	BASIC ASKS WHETHER NEW OR OLD PROGRAM IS TO BE RUN
NEW FILE NAME -- SAMPLE↵	BASIC ASKS FOR NEW FILENAME
READY	BASIC IS NOW READY TO RECEIVE STATEMENTS
10 FOR N = 1 TO 7↵	TYPE IN STATEMENTS
20 PRINT N, SQR(N)↵	

```
30 NEXT N↲
40 PRINT "DONE"↲
50 END↲
RUN↲                              RUN PROGRAM
SAMPLE   11:14   10/20/68
1            1
2            1.41421
3            1.73205
4            2
5            2.23607
6            2.44949
7            2.64575
DONE
↑C
.KJOB↲
CONFIRM:K↲
JOB 7, USER 27, 20 OFF TTY3 AT 0930 ON 29-OCT-68
FILES DELETED:  0, FILES SAVED:  0, RUNTIME 0 MIN, 01 SEC
.
```

1.7 ERRORS AND DEBUGGING

Occasionally the first run of a new problem will be free of errors and give the correct answers, but more commonly errors will be present and have to be corrected. Errors are of two types: (1) errors of form (or grammatical errors) which prevent the running of the program; (2) logical errors in the program which cause the computer 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 Section 2.8. Logical errors are 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 begin using arbitrary line numbers that are multiples of five or ten.

These corrections can be made at any time--whenever you notice them--either before or after a run. Since BASIC sorts lines out and arranges them in order, a line may be retyped out of sequence. Simply retype the offending line with its original line number.

1.7.1 Example of Finding and Correcting Program Errors

As with most problems in computing, we can best illustrate the process of finding the errors (or "bugs") in a program, and correcting (or "debugging") it, by an example. Let us consider the problem of finding that value of X between 0 and 3 for which the sine of X is a maximum, and ask the machine to print out this value of X and the value of its sine. Although we know that $\pi/2$ is the correct value we shall use the computer to test successive values of X from 0 to 3, first using intervals of .1, then of .01, and finally of .001. Thus, we shall ask the computer to find the sine of 0, of .1 of .2, of .3..., of 2.8, of 2.9, and of 3, and to determine which of these 31 values is the largest. It will do it by testing SIN(0) and SIN (.1) to see which is larger, and calling the larger of these two numbers M. Then it will pick the larger of M and SIN (.2) and call it M. This number will be checked against SIN (.3). Each time a larger value of M is found, the value of X is "remembered" in X0. When it finishes, M will have been assigned to the largest value. It will then repeat the search, this time checking the 301 numbers 0, .01, 02, .03, ..., 2.98, 2.99, and 3, finding the sine of each and checking to see which has the largest sine. At the end of each of these three searches, we want the computer to print three numbers: the value X0 which has the largest sine, the sine of that number, and the interval of search.

Before going to the Teletype, we write a program such as the following:

```
10 READ D
20 LET X0 = 0
30 FOR X = 0 TO 3 STEP D
40 IF SIN (X) <= M THEN 100
50 LET X0 = X
60 LET M = SIN (X0)
70 PRINT X0, X, D
80 NEXT X0
90 GO TO 20
100 DATA .1, .01, .001
110 END
```

We shall list the entire sequence on the Teletype and make explanatory comments on the right side.

```
NEW OR OLD--NEW ↲
NEW FILE NAME--MAXSIN ↲
READY
10 READ D↲
20 LWR X0 = 0 ↲
30 FOR X = 0 TO 3 STEP D↲
40 IF SINE \(X) <= M THEN 100↲
50 LET X0=X ↲
60 LET M = SIN(X) ↲
70 PRINT XO, X, D ↲
80 NEXT T \X0 ↲
90 GO TO 20 ↲
20 LET X0=0 ↲
100 DATA .1, .01, .001↲
110 END↲
RUN ↲

MAXSIN          11:35          10/20/68
```

Notice the use of the RUBOUT key (echoes as a \) to erase a character in line 40, which should have started IF SIN (X) etc., and in line 80.

After typing line 90, we notice that LET was mistyped in line 20, so we retype it, this time correctly.

After receiving the first error message, we inspect line 70 and find that we used XO for a variable instead of X0. The next two error messages relate to lines 30 and 80 having mixed variables. This is corrected by changing line 80.

```
ILLEGAL VARIABLE IN 70
NEXT WITHOUT FOR IN 80
FOR WITHOUT NEXT IN 30

70 PRINT X0, X, D↲
40 IF SIN(X) <= M THEN 80↲
80 NEXT X ↲
RUN↲

MAXSIN          11:36          10/20/68

0.100000        0.100000        0.1
0.200000        0.200000        0.1
0.300000
```

We make both of these changes by retyping lines 70 and 80. In looking over the program, we also notice that the IF-THEN statement in 40 directed the computer to a DATA statement and not to line 80 where it should go.
This is obviously incorrect. We are having every value of X printed, so we direct the machine to cease operations by typing S even while it is running. Note that the 'S' does not print. We notice that SIN(0) is compared with M on the first time through the loop, but we had assigned a value to X0 but not to M. However, we recall that all variables are set equal to zero before a RUN so that line 20 is unnecessary.

```
20 ↲
RUN ↲

MAXSIN          11:37          10/20/67

UNDEFINED LINE NUMBER 20 IN 90

90 GO TO 10 ↲
RUN ↲

MAXSIN          11:43          10/20/67

0.1             0.1             0.1
0.2             0.2             0.1
0.3
```

Of course, line 90 sent us back to line 20 to repeat the operation and not back to line 10 to pick up a new value for D. We retype line 90 and then type RUN again.

We are about to print out the same table as before. It is printing out X0, the current value of X, and the interval size each time that it goes through the loop.

```
70 ↵
85 PRINT X0, M, D↵
5 PRINT "X VALUE", "SIN", RESOLUTION"↵
RUN ↵
MAXSIN              11:44              10/20/67
```

We fix this by moving the PRINT statement outside the loop. Typing 70 deletes that line, and line 85 is outside of the loop. We also realize that we want M printed and not X. We also decide to put in headings for our columns by a PRINT statement.

```
ILLEGAL VARIABLE IN 5
```

There is an error in our PRINT statement: no left quotation mark for the third item.

```
5 PRINT "X VALUE", "SINE", "RESOLUTION"↵
RUN ↵
MAXSIN              11:47              10/20/67
```

Retype line 5, with all of the required quotation marks.

```
X VALUE             SINE            RESOLUTION
1.60000             0.999574          0.100000
1.57000             1.000000          0.01000
1.57099             1.000000          0.00100
OUT OF DATA IN 10
```

These are the desired results. Of the 31 numbers (0, .1, .2, .3,...,2.8, 2.9, 3) it is 1.6 which has the largest sine, namely .999574, similarly for finer subdivisions.

```
LIST↵
MAXSIN              11:48              10/20/67
5 PRINT "X VALUE", "SINE", "RESOLUTION"
10 READ D
30 FOR X = 0 TO 3 STEP D
40 IF SIN(X) <= M THEN 80
50 LET X0=X
60 LET M = SIN(X)
80 NEXT X
85 PRINT X0, M, D
90 GO TO 10
100 DATA .1, .01, .001
110 END
```

Having changed so many parts of the program, we ask for a list of the corrected program.

```
READY
SAVE↵
READY
```

The program is saved for later use.

A PRINT statement is inserted to check on the machine computations. For example, if we were to check M, we could have inserted 65 PRINT M, and seen the values.

1.8 SUMMARY OF ELEMENTARY BASIC STATEMENTS

In this section we shall give a short description of each of the types of BASIC statements discussed earlier in this chapter and add one statement to our list. In each form, we shall assume a line number, and use brackets to denote a general type. Thus, [variable] refers to any variable, which is a single letter, possibly followed by a single digit.

1.8.1 LET Statement

This statement is not of algebraic equality, but a command to the computer to perform certain computations and assign the answer to a certain variable. Each LET statement is of the form: LET [variable] = [formula] . More generally several variables may be assigned the same value by a single LET statement. Examples of assigning a value to a single variable are given in the following two statements:

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

Examples of assigning a value to more than one variable are given in the following statements:

50 LET X = Y3 = A(3,1) = 1	The variables X, Y3, and A(3,1) are assigned the value 1.
90 LET W = Z= 3*X-4*X ↑ 2	The variables W and Z are assigned the value 3*X-4*X ↑ 2

1.8.2 READ and DATA Statements

We use a READ statement to assign to the listed variable values obtained from a DATA statement. Neither statement is used without the other. 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, the computer takes all of the DATA statements in the order 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 the program is assumed to be done and we get an OUT OF DATA message.

Since we have to read in data before we can work with it, 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, 3.1415927

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.

1.8.3 PRINT Statement

The common uses of the PRINT statement are: (1) to print out the result of some computations, (2) to print out verbatim a message included in the program, (3) a combination of the two, and (4) to skip a line.

a. Examples of type (1):

```
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, a few spaces to the right of that number, its square root. The second will print five different numbers:

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

The computer will compute the two formulas and print up to five numbers per line in this format.

b. Examples of type (2):

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

Both have been encountered in the sample programs. The first prints that sample 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 -- as seen in MAXSIN.

c. Examples of type (3):

```
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.

d. Example of type (4):

250 PRINT

The computer will advance the paper one line when it encounters this command.

1.8.4 GO TO Statement

An example of requesting a different order of commands occurs in the MAXSIN problem where the computer has printed out X0, M, and D in line 85. To go through the same process for a different value of D, we directed the computer to revert to line 10 with a GO TO statement. Each is in the form of GO TO [line number].

Example: 150 GO TO 75

1.8.5 IF -- THEN Statement

To jump a normal sequence of commands, we use an IF -- THEN statement, sometimes called a conditional GO TO statement, such as line 40 of MAXSIN.

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 skips to line 80 if so. The second asks if G is equal to 0, and skips to line 65 if so. In each case, if the answer to the question is no, the computer will go to the next line.

1.8.6 ON...GO TO Statement

The IF--THEN--instruction allows a 2-way fork in a program. ON allows a many-way switch. For example:

80 ON X GO TO 100, 200, 150

This causes the following:

If X = 1, the program goes to line 100,

If X = 2, the program goes to line 200,

If X = 3, the program goes to line 150

More generally, in place of X any formula may occur, and there may be any number of line numbers in the instruction, as long as it fits on a single line. The value of the formula is computed and its integer part is taken. If this is 1, the program transfers to the line whose number is first on the list; if it is 2, to the second one, etc. If the integer part of the formula is below 1, or larger than the number of line numbers listed, an error message is printed. To increase the similarity between the ON and IF-THEN instructions, the instruction

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

may also be written as

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

Conversely, "THEN" may be used in an "ON" statement.

1.8.7 FOR and NEXT Statements

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 +1 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]
```

Examples:

```
30  FOR X = 0 TO 3 STEP D
80  NEXT X

120 FOR X4 = (17 + COX(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 variable (D), 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 the next example, the successive values of X will be 8, 7, 6, 5, 4, 3. In the last example, 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 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 the computer will proceed to the statement immediately following the corresponding NEXT statement.

1.8.8 DIM Statement

To enter a list or a table with a subscript greater than 10, use a DIM statement to retain sufficient space.

Examples:

```
20 DIM H(35)
35 DIM 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.

1.8.9 END Statement

Every program must have an END statement, and it must be the statement with the highest line number in the program.

Example:

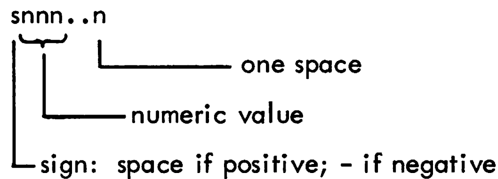
```
999 END
```

CHAPTER 2

ADVANCED BASIC STATEMENTS

2.1 MORE ABOUT THE PRINT STATEMENT

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 fifteen spaces each. 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. 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:



If you were to type the program

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

the Teletype would print 1 at the beginning of a line, 2 at the beginning of the next line, finally printing 15 on the fifteenth line. But, by changing line 20 to read

```
20 PRINT I,↵
```

the number is printed in the zones, reading

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

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

```
20 PRINT I;↵
```

and the result would be printed

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

A label inside quotation marks is printed as it appears and the end of a PRINT statement signals a new line, unless a comma or semicolon is the last symbol.

Thus, the instruction

```
50 PRINT X, Y↵
```

will print two numbers and then return to the next line, while

```
50 PRINT X, Y,↵
```

will print these two values and no return -- the next number to be printed will appear in the third zone, after the values of X and Y in the first two zones.

Since the end of a PRINT statement signals a new line

```
250 PRINT↵
```

will cause the Teletype to advance the paper one line, put a blank line for vertical spacing of your results, or complete a partially filled line.

```
 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, the Teletype 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 directs the Teletype, after printing the B(1,1) value corresponding to M = 1, to start a new line and after printing the value of B(2,2) corresponding to M = 2, etc.

The instructions

```
50 PRINT "TIME -"; "SHAR"; "ING";↵
51 PRINT " ON"; " THE"; " PDP-10"↵
```

will result in the printing of

TIME-SHARING ON THE PDP-10

Formatting of output can be controlled even further by use of the function TAB, in the form TAB(n), where n is the desired print position (0 through 74).

Insertion of TAB(17) causes the Teletype to move to column 17, as if a tab had been set there. For this purpose the positions on a line are numbered from 0 through 74, and 75 is assumed to be the 0 position again.

More precisely, TAB may contain any formula as its argument. The value of the formula is computed, and its integer part is taken. This in turn is treated modulo 75, to obtain a value from 0 through 74, as indicated above. The Teletype is then moved forward to this position (unless it has already passed this position, in which case the TAB is ignored).

For example, inserting the following line in a loop:

```
55 PRINT X;  TAB(12);  Y;  TAB(27);  Z ↲
```

will cause the X-value to start in column 0, the Y-values in column 12 and the Z-values in column 27.

The following rules are to interprete your printed results:

a. If a number is an integer, the decimal point is not printed. If the integer contains more than eight digits, it will be printed in the format

For example, it will take 32,437,580,259 and write it as 3.24376E+10.

b. For any decimal number, no more than six significant digits are printed.

c. For a number less than 0.1, the E notation is used unless the entire significant part of the number can be printed as a 6-digit decimal number. Thus, 0.03456 means that the number is exactly .0345600000, while 3.45600E-2 means that the number has been rounded to .0345600.

d. Trailing zeros 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 30↲
20 PRINT 2↑ N;↲
30 NEXT N↲
40 END↲
RUN↲

POWERS              11:54              10/20/68
```

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 131072 262144 524288
1048576 2097152 4194304 8388608 16777216 33553332 67108864
1.34218 E+8 2.68435 E+8 5.36871 E+8 1.07374 E+9

2.2 INT, RND, AND SGN FUNCTIONS, AND THE DEF STATEMENT

Five functions were listed in Section 1.3 but not described. We will discuss INT, RND and SGN here and leave NUM and DET until the MAT section (Section 2.6).

2.2.1 The INT (Integer) Function

The INT function appears in algebraic computation as [x], and gives the greatest integer under x. Thus INT (2.35) = 2, INT (-2.35) = -3, and INT (12) = 12. One use of the INT function is to round numbers 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 (3.4) = 3. It can also be used to round to any specific number of decimal places. For example, INT (10*X ↑ 2 + .5) / 10 ↑ 2 will round X correct to two decimal places, and INT (X*10 ↑ D + .5)/10 ↑ D round X correct to D decimal places.

2.2.2 The RND (Random Number Generating) Function

The function RND produces a random number between 0 and 1. RND does not require an argument.

If we want the first twenty random numbers, we write the program below and we get twenty 6-digit decimals.

```
10 FOR L = 1 TO 20↲
20 PRINT RND,↲
30 NEXT L↲
40 END↲
RUN↲

RNDNOS          13:24          10/20/68

0.406533        0.88445     0.681969        0.939462        0.253358
0.863799        0.880238    0.638311        0.602898        0.990032
0.570427        0.897931    0.628126        0.613262        0.303217
5.00548 E-2     0.393226    0.680219        0.632246        0.668218

RUN↲
```

```
RNDNOS          13:25          10/20/68

0.406533     0.88445     0.681969     0.939462     0.253358
0.863799     etc.
```

The second RUN gave exactly the same "random" numbers as the first RUN to facilitate the debugging of programs. If we want twenty random 1-digit integers, we could change line 20 to read

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

and we would then obtain

```
RNDNOS          13:26          10/20/68

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

To vary the type of random numbers (20 random numbers ranging from 1 to 9 inclusive) change line 20 as follows:

```
20 PRINT INT (9*RND + 1); ↵
RUN ↵

RNDNOS          13:28          10/20/68

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

to obtain random numbers which are integers from 5 to 24 inclusive, change line 20 to

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

RNDNOS          13:28          10/20/68

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

If random numbers are to be chosen from the A integers of which B is the smallest, call for INT (A*RND +B).

2.2.3 The RANDOMIZE Statement

As noted when we ran the first program of this section twice, we got the same numbers in the same order each time. However, we can get a different set by use of the instruction RANDOMIZE as in the following program.

```
5 RANDOMIZE ↵
10 FOR L = 1 TO 20 ↵
20 PRINT INT(10*RND); ↵
30 NEXT L ↵
40 END ↵
RUN ↵

RNDNOS          13:32          10/20/68

1  9  4  2  1  1  6  6  3  8  4  9  8  6  5  8  6  2  6  0

RUN ↵

RNDNOS          13:33          10/20/68

1  1  4  6  6  6  0  5  3  8  4  0  8  1  0  5  1  8  0  1
```

RANDOMIZE (RANDOM) resets the numbers in a random way. For example, if this is the first instruction in a program using random numbers, then repeated RUNs of the program will produce different results. If the instruction is absent, then the "official list" of random numbers is obtained in the usual order. It is suggested that a simulated model should be debugged without this instruction, so that one always obtains the same random numbers in test runs. After the program is debugged, one inserts

```
1  RANDOM
```

before starting production runs.

2.2.4 The SGN (Sign) Function

The SGN function is one which 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. For example, the statement 50 ON SGN(X) + 2 GO TO 100, 200, 300 will transfer to 100 if $X < 0$, to 200 if $X = 0$, and to 300 if $X > 0$.

2.2.5 The DEF (Define User Function) and FNEND (Function End) Statements

Define any function you expect to use repeatedly by a DEF statement. The name of the defined function must be three letters, the first two of which are FN. Hence, you may define up to 26 functions, e.g., FNA, FNB, etc. Where you frequently need the function $e^{-x^2} + 5$, introduce the function by the line

```
30 DEF FNE(X) = EXP(-X ↑ 2 + 5)
```

and later on call for various values of the function by FNE(.1), FNE(3.45), FNE(A+2), etc. Such definition can be a great time saver when you want values of some function for a number of different values of the variable.

The DEF statement may occur anywhere in the program, and the expression to the right of the equal sign may be any formula which can be fitted onto one line. It may include any combination of other functions, including ones defined by different DEF statements, and it can involve other variables besides those denoting the argument of the function.

Each function defined may have zero, one, two, or more variables. For example:

```
10 DEF FNB(X,Y) = 3*X*Y - Y↑3
105 DEF FNC(X,Y,Z,W) = FNB(X,Y)/FNB(Z,W)
530 DEF FNA = 3.1416*R↑2
```

In the definition of "FNA" the current value of R is used when FNA occurs. Similarly, if FNR is defined by

```
70 DEF FNR(X) = SQR(2 + LOG(X) - EXP(Y*Z)*(X + SIN(2*Z) ) )
```

you can ask for FNR(2.7), and give new values to Y and Z before the next use of FNR.

The method of having multiple line DEFs is illustrated by the 'max' function. In this the possibility of using 'IF...THEN' as part of the definition is a great help:

```
10 DEF FNM(X,Y)
20 LET FNM = X
30 IF Y <= X THEN 50
40 LET FNM = Y
50 FNEND
```

The absence of the '=' sign in line 10 indicates that this is a multiple line DEF. FNEND in line 50 terminates the definition. The expression 'FNM' without an argument serves as a temporary variable for the computation of the function value. The following example defines N-factorial:

```
10 DEF FNF(N)
20 LET FNF = 1
30 FOR K = 1 TO N
40 LET FNF = K * FNF
50 NEXT K
60 FNEND
```

Any variable which is not an argument of FN_ in a DEF loop will have its current value in the program. Multiple line DEFs may not be nested and there must not be a transfer from inside the DEF to outside its range, or vice-versa.

2.3 SUBROUTINES: GOSUB AND RETURN STATEMENTS

When a particular part of a program is to be repeatedly performed, it is most efficiently programmed as a subroutine and entered with a GOSUB statement; the number is the line number of the first statement in the subroutine. For example,

```
90 GOSUB 210
```

directs the computer to jump to line 210, the first line of the subroutine. The last line of the subroutine should be a RETURN command to the earlier part of the program.

```
350 RETURN
```

will revert to the first line numbered greater than 90.

A program for determining the greatest common divisor (GCD) of three integers using the Euclidean Algorithm, illustrates the use of a subroutine. The first two numbers are selected in line 30 and 40 and their GCD is determined in the subroutine, lines 200 through 310. The GCD just found is called X in line 60, the third number 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.

A GOSUB inside a subroutine to perform another subroutine is called "nested GOSUBs." It is necessary that you exit from a subroutine only with a RETURN statement. 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 THEN 300↵
230 LET X = Y↵
240 LET Y = R↵
250 GO TO 200↵
300 LET G = Y↵
310 RETURN↵
320 END↵
RUN↵

GCD3NOS.        13:38        10/20/68
```

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

OUT OF DATA IN 20

2.4 INPUT STATEMENT

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 saves it on the storage device as a library program[1], and other persons use the program and supply their own data. Data may be entered by an INPUT statement, which acts as a READ but accepts numbers of alphanumeric data from the Teletype keyboard. For example, to supply values for X and Y into a program, type

```
40 INPUT X, Y ↵
```

prior to 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, presses the RETURN key, and BASIC goes on with the rest of the program. No number can be greater than 8 digits in length.

---

[1] See SAVE command, Section 2.9.

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. Therefore, INPUT should be used only when small amounts of data are to be entered, or when necessary during the running of the program.

2.5 STOP, REM (REMARKS), AND RESTORE STATEMENTS

Several other useful BASIC statements are STOP, REM and RESTORE.

2.5.1 STOP Statement

STOP is equivalent to GOTO xxxxx, where xxxxx is the line number of the END statement in the program. For example, the following two program portions are exactly equivalent.

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

2.5.2 REM (Remarks) Statement

REM provides a means for inserting explanatory remarks in the program. BASIC completely ignores the remainder of that line, allowing you to follow the REM with directions for using the program, with identifications of the parts of a long program, or with any other information. 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
```

There is a second method for adding comments to a program. Place an ' (apostrophe) at the end of the line, followed by a remark. Everything following the ' is ignored except when the line ends in a string (see Section 2.7).

2.5.3 RESTORE Statement

The RESTORE statement permits READing the data in the DATA statements of a program more than once. Whenever RESTORE is encountered in a program, BASIC restores the data block pointer to the first number. A subsequent READ statement will then start reading the data all over again. However, if the desired data is 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 block to its original state, and reads the data again. Note the use of line 570 (READ X) 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
590  READ X
      . . . . .
700 DATA. . . . .
710 DATA. . . . .
```

2.6 MATRICES

There is a special set of thirteen instructions for matrix computations, identified by the starting word 'MAT'. They are

MAT READ a, b, c	Read the three matrices, their dimensions having been previously specified.
MAT c = ZER	Fill out c with zeroes.
MAT c = CON	Fill out c with ones.
MAT c = IDN	Set up c as an identity matrix.

MAT PRINT a, b, c	Print the three matrices (semicolons can be used immediately following any matrix which you desire to be printed in a closely packed format).
MAT INPUT v	Calls for the input of a vector.
MAT b = a	Set the matrix b equal to the matrix a.
MAT c = a + b	Add the two matrices a and b.
MAT c = a - b	Subtract the matrix b from the matrix a.
MAT c = a * b	Multiply the matrix a by the number b.
MAT c = TRN(a)	Transpose the matrix a.
MAT c = (k) * a	Multiply the matrix a by the number k. The number k, which must be in parentheses may also be given by a formula.
MAT c = INV (a)	Invert the matrix a.

2.6.1 MAT Statement Conventions

The following convention has been adopted for MAT: while every vector has a component 0, and every matrix has a row 0 and a column 0, the MAT instructions ignore these. Thus if in a MAT instruction we have a matrix of dimension M-by-N, the rows are numbered 1, 2, ..., M, and the columns 1, 2, ..., N.

The DIM statement may simply indicate what the maximum dimension is to be. Thus, if we write

```
DIM M(20,35)
```

then M may have up to 20 rows and up to 35 columns. This statement is to save enough space for the matrix, and hence, the only care at this point is that the dimensions declared are large enough to accommodate the matrix. However, in the absence of DIM statements all vectors may have up to 10 components and matrices up to 10 rows and 10 columns. This is to say that in the absence of DIM statements this much space is automatically saved for vectors and matrices on their appearance in the program. The actual dimension of a matrix may be determined either when it is first set up (by a DIM statement) or when it is computed. Thus

```
10  DIM M(20,7)
    - - - -
50  MAT READ M
```

will read a 20-by-7 matrix for M, while

```
50  MAT READ M(17,30)
```

will read a 17-by-30 matrix for M, provided sufficient space has been saved for it by writing, for example,

```
10  DIM M(20,35).
```

2.6.2 MAT c = ZER, MAT c = CON, MAT c = IDN

The three instructions

```
MAT  M = ZER
MAT  M = CON
MAT  M = IDN,
```

which set up a matrix M with all components zero, all components equal to one, and as an identity matrix, respectively, act like MAT READ as far as the dimension of the resulting matrix is concerned. For example,

```
MAT  M = CON(7,3)
```

sets up a 7-by-3 matrix with 1 in every component, while

```
MAT  M = CON
```

sets up a matrix, with ones in every component, and of dimension 10-by-10 unless previously dimensioned otherwise. It should be noted, however, that these instructions have no effect on row and column zero. Thus

```
10  DIM M (20,7)
20  MAT READ M(7,3)
. . . . . . .
35  MAT M = CON
. . . . . . .
70  MAT M = ZER (15,7)
. . . . . . .
90  MAT M = ZER (16,10)
```

will first read in a 7-by-3 matrix for M. Then it will set up a 7-by-3 matrix of all 1s for M (the actual dimension having been set up as 7-by-3 in line 20.) Next it will set up M as a 15-by-7 all zero matrix.

(Note that although this is larger than the previous M, it is within the limits set in 10.) But it will result in an error message in line 90. The limit set in line 10 is (20 + 1) x (7 + 1) = 168 components, and in 90 we are calling for (16 + 1) x (10 + 1) = 187 components. Thus, although the zero rows and columns are ignored in MAT instructions they play a role in determining dimension limits. So, for example

```
90  MAT M = ZER(25,5)
```

would not yield an error message.

It, perhaps, should be noted that an instruction such as MAT READ M(2,2) which sets up a matrix and which as we have said ignores the zero row and column does however affect the zero row and column. The redimensioning which may be implicit in an instruction causes the relocation of some numbers and so they may not appear subsequently in the same place. Thus even if we have first LET M(1,0) = M(2,0) = 1, and then MAT READ M(2,2) the values of M(1,0) and M(2,0) will now be 0. Thus, when using MAT instructions, it is best not to use row and column zero.

2.6.3 MAT PRINT a, b, c

The instruction

```
MAT PRINT A, B ; C
```

will cause the three matrices to be printed with A and C in the normal format (i.e., with five components to a line and starting each new row on a new line) and B closely packed.

Vectors may be used in place of matrices, as long as the above rules are observed. Since a vector like V(I) is treated as a column vector by BASIC, a row vector has to be introduced as a matrix that has only one row, namely row 1. Thus

```
DIM X(7), Y(0,5)
```

introduces a 7-component column vector and a 5-component row vector.

If V is a vector then

```
MAT PRINT V
```

will print the vector V as a column vector.

```
MAT PRINT V,
```

will print V as a row vector, five numbers to the line, while

```
MAT PRINT V;
```

will print V as a row vector, closely packed.

2.6.4 MAT INPUT V and the NUM Function

The instruction

```
MAT INPUT V
```

will call for the input of a vector. The number of components in the vector need not be specified. Normally the input is limited by having to be typed on one line. However by ending the line of input with & (before carriage return) the machine will ask for more input on the next line. Note that, although the number of components need not be specified, if we wish to input more than 10 numbers we must save sufficient space with a DIM statement. After the input the function NUM will equal the number of components and V(1), V(2), ..., V(NUM) will be the numbers inputted. This allows variable length input. For example

```
 5 LET S = 0
10 MAT INPUT V
20 LET N = NUM
30 IF N = 0 THEN 99
40 FOR I = 1 TO N
45 LET S = S + V(I)
50 NEXT I
60 PRINT S/N
70 GO TO 5
99 END
```

allows the user to type in sets of numbers, which are averaged. The program takes advantage of the fact that zero numbers may be inputted, and uses this as a signal to stop. Thus, the user can stop by simply pushing "carriage return" on an input request.

2.6.5 MAT b = a

This sets b up to be the same as a and in doing so dimensions b to be the same as a, provided sufficient space has been saved for b.

2.6.6 MAT c = a + b and MAT c = a-b

For these to be legal, a and b must have the same dimensions, and enough space must be saved for c. These statements cause c to assume the same dimensions as a and b. Instructions such as MAT A = A ± B are legal - the indicated operation is performed and the answer stored in A. Only a single arithmetic operation is allowed so MAT D = A + B - C is illegal but may be achieved with two MAT instructions.

2.6.7 MAT c = a * b

For this to be legal, it is necessary that the number of columns in a be equal to the number of rows in b. For example, if matrix A has dimension L-by-M and matrix B has dimension M-by-N then C = A * B will have dimension L-by-N. It should be noted that while MAT A = A + B may be legal, MAT A = A * B will result in errors because in multiplying two matrices, we will destroy components which would be needed to complete the computation. MAT B = A * A is, of course, legal provided A is a 'square' matrix.

2.6.8 MAT c = TRN(a)

This lets c be the transpose of the matrix a. Thus if matrix A is an M-by-N, matrix C will be an N-by-M matrix.

2.6.9 MAT c = (k) * a

This lets c be the matrix a multiplied by the number k (i.e., each component of a is multiplied by k to form the components of c). The number k, which must be in parentheses, may be replaced by a formula. MAT A = (K) * A is legal.

2.6.10 MAT c = INV(a) and the DET Function

This lets c be the inverse of a. (a must, of course, be a 'square' matrix.) The function DET is available after the execution of the inversion, and will equal the determinant of a. This enables the user to decide whether the determinant was large enough for the inverse to be meaningful. In particular, attempting to invert a singular matrix will not cause the program to stop, but DET is set equal to 0. Of course, the user may actually want the determinant of a matrix; he may obtain this by inverting the matrix and then seeing what value DET has.

2.6.11 Examples of Matrix Programs

We close this section with two illustrations of programs involving matrices. The first one reads in A and B in line 30 and in so doing sets up the correct dimensions. Then, in line 40, A + A is computed and the answer is called C - this automatically dimensions C to be the same as A. Note that the data in line 90 results in A being 2-by-3 and B being 3-by-3. Both MAT PRINT formats are illustrated, and one method of labeling a matrix print is shown.

```
10 DIM A(20,20),B(20,20),C(20,20)↵
20 READ M,N↵
30 MAT READ A(M,N),B(N,N)↵
40 MAT C = A + A↵
50 MAT PRINT C;↵
60 MAT C = A * B↵
70 PRINT↵
75 PRINT "A * B = ",↵
80 MAT PRINT C↵
90 DATA 2,3↵
91 DATA 1,2,3↵
92 DATA 4,5,6↵
93 DATA 1,0,-1↵
94 DATA 0,-1,-1↵
95 DATA -1,0,0↵
99 END↵
RUN↵

MATRIX              13:48              10/20/68
_______________________________________________

2       4       6
8       10      12

A*B =

-2          -2          -3
-2          -5          -9
```

The second example inverts an n-by-n Hilbert Matrix

```
1       1/2       1/3 . . .     1/n
1/2     1/3       1/4 . . .     1/n + 1
1/3     1/4       1/5 . . .     1/n + 2
 .       .         . . . .
 .       .         . . . .
 .       .         . . . .
1/n     1/n + 1    1/n + 2      1/2n-1
```

Ordinary BASIC instructions are used to set up the matrix in lines 50 to 90. Note that this occurs after correct dimensions have been declared. Then a single instruction results in the computation of the inverse, and one more instruction prints it. The fact that the function DET is available after an inversion is also taken advantage of in line 130 to print the value of the determinant of A. In this example we have supplied 4 for N in the DATA statement and have made a run for this case.

```
  5 REM THIS PROGRAM INVERTS AN N-BY-N HILBERT MATRIX
 10 DIM A(20,20),B(20,20)
 20 READ N
 30 MAT A = CON(N,N)
 50 FOR I = 1 TO N
 60 FOR J = 1 TO N
 70 LET A(I,J) = 1/(I + J - 1)
 80 NEXT J
 90 NEXT I
100 MAT B = INV(A)
115 PRINT "INV(A) = ",
120 MAT PRINT B;
125 PRINT
130 PRINT "DETERMINANT OF A = " DET
190 DATA 4
199 END
RUN

HILMAT              13:52              10/20/68

INV(A) =

 16.0001  -120.001   240.003  -140.002
-120.001   1200.01  -2700.03   1680.02
 240.003  -2700.03   6480.08  -4200.05
-140.002   1680.02  -4200.05   2800.03

DETERMINANT OF A = 1.65342 E-7
```

A 20-by-20 matrix is inverted in about .5 seconds. However, the reader is warned that beyond n = 7 the Hilbert matrix cannot be inverted because of severe round-off-errors.

2.6.12 Simulation of n-Dimensional Arrays

Although it is not possible to create n-dimensional arrays in BASIC, the method outlined below will simulate them. The example is of a 3-dimensional array but it has been written in such a way that it could be changed to 4 or higher dimensions easily. We use the fact that functions can have any number of variables and we set up a 1-to-1 correspondence between the components of the array and the components of a vector which will equal the product of the dimensions of the array. For

example, if the array has dimensions 2, 3, 5 then the vector will have 30 components. A multiple line DEF could be used in place of the simple DEF in line 30 if the user wished to include error messages. The printout is in the form of two 3-by-5 matrices.

```
 10 DIM V(1000)↵
 20 MAT READ D(3)↵
 30 DEF FNA(I,J,K) = ( ( I-1) * D(2) + (J-1) ) *D(3) + K↵
 50 FOR I = 1 TO D (1)↵
 60 FOR J= 1 TO D (2)↵
 70 FOR K = 1 TO D (3)↵
 80 LET V(FNA(I,J,K) ) = I + 2*J + K ↑ 2↵
 90 PRINT V(FNA(I,J,K) ),↵
100 NEXT K↵
110 NEXT J↵
112 PRINT↵
115 PRINT↵
120 NEXT I↵
900 DATA 2,3,5↵
999 END↵
 RUN↵

 3-ARRAY          08:07          10/27/68
 ________________________________________

 4        7        12       19       28
 6        9        14       21       30
 8        11       16       23       32

 5        8        13       20       29
 7        10       15       22       31
 9        12       17       24       33
```

2.7 ALPHANUMERIC INFORMATION (STRINGS)

Our discussion of BASIC in previous sections dealt only with numerical information. However, BASIC will also handle alphabetic or alphanumeric information. We define a string to be a sequence of characters, each of which is either a letter, a digit, a space, or some other printable character.

We may introduce variables for single strings and 'string' vectors (but not 'string' matrices). Any ordinary variable followed by a $ will stand for a string. For example A$ or C7$ can denote strings. A vector variable followed by $, e.g. V$ ( ), will denote a list of strings. Thus, V$ (7) is the 7th string in the list.

2.7.1 Reading and Printing Strings

First of all, strings may be read and printed. For example,

```
10  READ A$, B$, C$
20  PRINT C$; B$; A$
30  DATA ING, SHAR, TIME-
40  END
```

will print the word "time-sharing." Note that the effect of the semicolon in the PRINT statement is consistent with that discussed in the section on PRINT, i.e. with alphanumeric output the semicolon causes close packing whether that output is in quotes or is the value of a variable. Commas and TABs may be used as in any other PRINT statement. The loop

```
70  FOR I = 1 TO 12
80  READ M$(I)
90  NEXT I
```

will read a list of 12 strings.

In place of the READ and PRINT, corresponding MAT instructions may be used for lists. For example, MAT PRINT M$; will cause the members of the list to be printed without spaces between them. We may also use INPUT or MAT INPUT. After a MAT INPUT the function NUM will equal the number of strings inputted.

As usual, lists are assumed to have no more than 10 elements, otherwise a DIM statement is required. The statement

```
10  DIM  M$(20)
```

saves room for twenty strings in the M$-list.

In the DATA statements, numbers and strings may be intermixed. Numbers will be assigned only to numerical variables, and strings only to string-variables. Strings in DATA statements are recognized by the fact that they start with a letter. If not, it must be enclosed in quotes. The same requirement holds for a string containing a comma. For example:

```
90  DATA 10, ABC, 5, "4FG", "SEPT. 22, 1968", 2
```

The only convention on INPUT is that a string containing a comma must be enclosed in quotes.

With a MAT INPUT a string containing a comma or an ampersand (&) must be enclosed in quotes. For example:

```
"MR. & MRS. SMITH", MR. JONES
```

is in correct format for a response to a MAT INPUT.

2.7.2 String Conventions

In the three ways of getting string information into a program (DATA, INPUT or MAT INPUT), leading blanks are ignored unless the string, including the blanks, is enclosed in quotes. Strings (in quotes) or string-variables may occur in LET and IF-THEN statements. The following two examples are self-explanatory:

```
10 LET Y$ = "YES"
20 IF Z7$ = "YES" THEN 200
```

The relation " < " is interpreted as "earlier in alphabetic order." The other relational symbols work in a similar manner. In any comparison, trailing blanks in a string are ignored. Thus,

```
"YES" = "YES  ".
```

We illustrate these possibilities by the following program which reads a list of strings, and alphabetizes them:

```
 10 DIM L$(50)
 20 READ N
 30 MAT READ L$ (N)
 40 FOR I = 1 TO N
 50 FOR J = 1 TO N-I
 60 IF L$(J) = L$(J + 1) THEN 100
 70 LET A$ = L$(J)
 80 LET L$(J) = L$(J + 1)
 90 LET L$(J + 1) = A$
100 NEXT J
110 NEXT I
120 MAT PRINT L$
900 DATA 5, ONE, TWO, THREE, FOUR, FIVE
999 END
```

If we omit the $ signs in this program, it serves to read a list of numbers and prints them in increasing order.

A rather common use is illustrated by the following:

```
330 PRINT "DO YOU WISH TO CONTINUE";
340 INPUT A$
350 IF A$ = "YES" THEN 10
360 STOP
```

2.7.3 Numeric and String DATA Blocks; RESTORE* RESTORE $ Statements

Numeric and string DATA are kept in two separate blocks, and these act independently of each other. Therefore, RESTORE will retain both the numerical data and the string data. RESTORE* will retain only the numerical data and RESTORES$ only the string data.

2.7.4 Accessing Individual Characters; the CHANGE Statement

In BASIC it is very easy to obtain the individual digits in a number by using the function INT. It is possible to obtain the individual characters in a string with the instruction CHANGE. The use of CHANGE is best illustrated with examples.

```
 5 DIM A(65)↵
10 READ A$↵
15 CHANGE A$ TO A↵
20 FOR I = 0 TO A(0)↵
25 PRINT A(I);↵
30 NEXT I↵
40 DATA ABCDEFGHIJKLMNOPQRSTUVWXYZ↵
45 END↵
RUN↵

CHANGE          13:55          10/20/68

26 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90
```

The instruction CHANGE A$ TO A in line 15 has caused the vector A to have as its zero component the number of characters in the string A$ and to have certain numbers in the other components. These numbers are the BASIC 'code' numbers for the characters appearing in the string (e.g., A(1) is 65 - the BASIC code number for A).

The BASIC code for the printable characters is

Character	BASIC Code No. (Decimal)	Character	BASIC Code No. (Decimal)
" "	32	"@"	64
"!"	33	"A"	65
"""	34	"B"	66
"#"	35	"C"	67
"$"	36	"D"	68
"%"	37	"E"	69
"&"	38	"F"	70
"'"	39	"G"	71
"("	40	"H"	72
")"	41	"I"	73
"*"	42	"J"	74
"+"	43	"K"	75
","	44	"L"	76
"-"	45	"M"	77
"."	46	"N"	78
"/"	47	"O"	79
"0"	48	"P"	80
"1"	49	"Q"	81
"2"	50	"R"	82
"3"	51	"S"	83
"4"	52	"T"	84
"5"	53	"U"	85
"6"	54	"V"	86
"7"	55	"W"	87
"8"	56	"X"	88
"9"	57	"Y"	89
":"	58	"Z"	90
";"	59	"["	91
"<"	60	"\"	92
"="	61	"]"	93
">"	62	"↑"	94
"?"	63		

Additional symbols useful on output are

← (backward arrow)	95
LF (line feed)	10
CR (carriage return)	13

This is not a complete list; there are 128 characters numbered 0 through 127. Some of these numbers duplicate the above (on some Teletypes); some are for Teletypes with upper and lower case letters, and some are useless.

The other use of CHANGE is illustrated by

```
10  FOR I = 0 TO 5
15  READ A(I)
20  NEXT I
25  DATA 5, 65, 66, 67, 68, 69
30 CHANGE A TO A$
35  PRINT A$
40  END
```

This will print ABCDE because the numbers 65 through 69 are the code numbers for A through E. Before CHANGE is used in the 'vector to string' direction, we must give the number of characters which are to be in the string as the zero component of the vector. Above, A(0) is read as 5. A final example:

```
 5  DIM V(128)
10  PRINT "WHAT DO YOU WANT THE VECTOR V TO BE";
20  MAT INPUT V
30  LET V(0) = NUM
40  CHANGE V TO A$
50  PRINT A$
60  GO TO 10
70  END
RUN

EXAMPLE              13:59              10/20/68

WHAT DO YOU WANT THE VECTOR V TO BE? 40,32,45,60,45,89,90
( -<-YZ
WHAT DO YOU WANT THE VECTOR V TO BE? 32,33,34,35,36,37,38,39,40,41,42,43&
? 44,45,46,47,48,49,50
! " # $ % & ' ( ) * + , - . / 012
WHAT DO YOU WANT THE VECTOR V TO BE? 4
```

Note that in this example we have used the availability of the function NUM after a MAT INPUT to find the number of characters in the string which is to result from line 40. Giving the input "4" on the last request obtains the response EOT (end of transmission), which turns off the Teletype.

2.8 DIAGNOSTIC MESSAGES

Most messages typed out by BASIC are self-explanatory. A full listing of these messages and their meanings will be made available in the near future.

2.9 EDIT AND CONTROL COMMANDS

Several commands for editing BASIC programs and for controlling their execution enable you to: delete lines, list your program, change or resequence line numbers with set increments, save programs on a file-structured storage device (disk or DECtape), 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
DELETE n	Delete line number n.
DELETE n_1,n_2	Delete line numbers n_1 through n_2.
LENGTH	Print length of source program (expressed as number of characters).
LIST	List program with heading.
LIST --n	List program with heading, beginning at line number n.
LIST --n_1,n_2	List program with heading, from line number n_1 through n_2.
LISTNH LISTNH --n LISTNH --n_1,n_2	Same as above, but with heading suppressed.
NEW	BASIC will ask for new program name and will check that it does not already exist.
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.
RENAME fname	Change name of program currently in user core.
REPLACE	Replace old file of current name with contents of user core.
RUN	Compile and run program currently in core.
SAVE	Save the contents of user core as file whose filename is current program name and whose extension is .BAS[1].
SAVE fname	Save user core as fname .BAS[1].
SCRATCH	Delete all program statements from user core.
REQUENCE n	Change line numbers to n, n + 10, ...
RESEQUENCE n,k	Change line numbers to n, n + k, ...
RESEQUENCE n, f, k	Change line numbers from line f upward to n, n + k, ...

[1]SAVE commands will not overwrite an existing file of the same name (use REPLACE, instead).

Command	Action
SYSTEM	Exit to Monitor.
WEAVE fname	Read program statements from the file named fname.BAS (existing statements in user core are replaced by new statements having same line numbers).
↑C	To stop a running program, type ↑C followed by REENTER.

APPENDIX A

SUMMARY OF BASIC STATEMENTS

Elementary BASIC Statements

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

Statement	Description
LET [variable] = [formula]	Assign the value of the formula to the specified variable.
DATA [data list] READ [sequence of variables]	DATA statements are used to supply one or more numbers or alphanumeric strings to be accessed by READ statements. READ statements, in turn, assign the next available datum in the DATA string to the variables listed. Numeric and alphanumeric data are kept in separate tables; however, they both may be entered in the same DATA statement.
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] {THEN / GO TO} [line number]	If the stated relationship is true, then transfer control to the line number specified; if not, continue in sequence.
ON [x] {GO TO / THEN} [line number$_1$,] [line number$_2$,] .... [line number$_n$]	If the integer portion of x = 1, transfer control to line number$_1$, if x = 2, to line number$_2$, etc. [x] may be a formula.
FOR [variable] = [formula$_1$] TO [formula$_2$] STEP [formula$_3$] NEXT [variable]	Used for looping repetitively through a series of steps. The FOR statement initializes the variable to the value of formula$_1$ and then performs the following steps until the NEXT statement is encountered. The NEXT statement increments the variable by the value of formula$_3$ (if omitted, the increment value is assumed to be +1); the resultant value is then compared to the value of formula$_2$. If variable < formula$_2$, control is sent back to the step following the FOR statement and the sequence of steps is repeated; eventually, when variable $\geq$ formula$_3$, control continues in sequence at the step following NEXT.
DIM [variable] (subscript)	Enables the user to enter a table or array with a subscript greater than 10 (i.e., more than 10 items).
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 (↑), Advanced BASIC includes the following elementary functions:

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

Advanced BASIC Statements

Statement	Explanation
GOSUB [line number]	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).
Subroutine { [line number] . . . . . RETURN	
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).

Matrix Instructions

(NOTE: The word "vector" may be substituted for the word "matrix" in the following explanations.)

Instruction	Explanation
MAT READ a, b, c	Read the three matrices, their dimensions having been previously specified.
MAT c = ZER	Fill out c with zeroes.
MAT c = CON	Fill out c with ones.
MAT c = IDN	Set up c as an identity matrix.
MAT PRINT a, b, c	Print the three matrices.
MAT INPUT v	Input a vector.
MAT b = a	Set matrix b = matrix a.
MAT c = a + b	Add the two matrices, a and b.
MAT c = a - b	Subtract matrix b from matrix a.
MAT c = a * b	Multiply matrix a by matrix b.
MAT c = TRN(a)	Transpose matrix a.
MAT c = (k) * a	Multiply matrix a by the number k. (k, which must be in parentheses, may also be given by a formula.)
MAT c = INV(a)	Invert matrix a.

STOP	Equivalent to GO TO [line number of END statement].
REM	Permits typing of remarks within the program. The insertion of short comments following any BASIC statement is accomplished by preceding such comments with an apostrophe(').
RESTORE	Sets pointer back to beginning of string of DATA values.

FORMULAS: Some advanced functions include the following:

INT (x)	Find the greatest integer not greater than x.
RND	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 statement RANDOMIZE can be used to cause the generation of new sets of random numbers.
SGN (x)	Assign a value of 1 if x is positive; 0 if x is 0; or -1 if x is negative.

Two special functions are used with matrix computations.

(NUM)	Equals number of components following an INPUT.
DET	Equals the determinant of a matrix after inversion.

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

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

NOTE that DEFine statements may be extended onto more than one line; all other statements are restricted to a single line.

ADVANCED BASIC for PDP-10
DEC-10-KJZA-D

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively, we need user feedback: your critical evaluation of this manual and the DEC products described.

Please comment on this publication. For example, in your judgment, is it complete, accurate, well-organized, well-written, usable, etc? ______________________________

Did you find this manual easy to use? ______________________________

What is the most serious fault in this manual? ______________________________

What single feature did you like best in this manual? ______________________________

Did you find errors in this manual? Please describe. ______________________________

Please describe your position. ______________________________

Name ____________________ Organization ____________________

Street ____________________ State ____________________ Zip __________

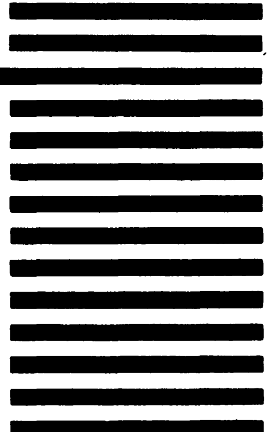
Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:



digital

Digital Equipment Corporation
Software Quality Control
Building 12
146 Main Street
Maynard, Mass. 01754

digital

DIGITAL EQUIPMENT CORPORATION □ MAYNARD, MASSACHUSETTS

Printed in U.S.A.