GENERAL ELECTRIC
COMPUTERS

'BASIC' LANGUAGE

REFERENCE MANUAL

GENERAL ELECTRIC

IPC-202026

'BASIC'
LANGUAGE

REFERENCE MANUAL

June 1965

Rev. September 1966

GENERAL ELECTRIC

INFORMATION SYSTEMS DIVISION

PREFACE

This manual is a reference for the BASIC language used with the General Electric Computer Time-Sharing Service.

The development of the BASIC language and the original version of this manual were supported by the National Science Foundation under the terms of a grant to Dartmouth College. Under this grant, Dartmouth College, under the direction of Professors John G. Kemeny and Thomas E. Kurtz, developed the BASIC language compiler and the necessary executive routines for the GE-235 and the DATANET-30*.

The printing of this manual by General Electric does not necessarily constitute endorsement of General Electric products by Dartmouth College.

This edition does not obsolete the previous edition dated May 1966. It does, however, contain several minor revisions which are indicated by a bar in the margin opposite the change.

For comments about this publications use the Reader's Comments sheet in the back of the manual, or address comments directly to Technical Publications, General Electric Company, 2725 North Central Avenue, Phoenix, Arizona 85004.

* DATANET is a Reg. Trademark of the General Electric Company.

CONTENTS

APPENDIXES

1. WHAT IS A PROGRAM?

A program is a set of directions, or a recipe, that is used to tell a computer how to provide an answer to some problem. It usually starts with the given data as the ingredients, contains a set of instructions to be performed or carried out in a certain order, and ends up with a set of answers as the cake. And, as with ordinary cakes, if you make a mistake in your program, you will end up with something else--perhaps hash !

Any program must fulfill two requirements before it can be carried out. The first is that it must be presented in a language that is understood by the "computer." If the program is a set of instructions for solving a system of linear equations and the "computer" is an English-speaking person, the program will be presented in some combination of mathematical notation and English. If the "computer" is a French-speaking person, the program must be in his language; and if the "computer" is a high-speed digital computer, the program must be presented in a language which the computer "understands."

The second requirement for all programs is that they must be completely and precisely stated. This requirement is crucial when dealing with a digital computer which has no ability to infer what you mean--it does what you tell it to do, not what you meant to tell it.

We are, of course, talking about programs which provide numerical answers to numerical problems. It is easy for a programmer to present a program in the English language, but such a program poses great difficulties for the computer because English is rich with ambiguities and redundancies. those qualities which make poetry possible, but computing impossible. Instead, you present your program in a language which resembles ordinary mathematical notation, which has a simple vocabulary and grammar, and which permits a complete and precise specification of your program. The language you will use is BASIC (Beginner's All-purpose Symbolic Instruction Code) which is, at the same time, precise, simple, and easy to understand.

A first introduction to writing a BASIC program is given in Chapter 2. This chapter includes all that you will need to know to write a wide variety of useful and interesting programs. Chapters 3 and 4 deal with more advanced computer techniques, and the Appendices contain a variety of reference materials.

2. A BASIC PRIMER

2.1 AN EXAMPLE

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

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

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

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

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

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

Study this example carefully--in most cases the purpose of each line in the program is self-evident--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
```

We immediately observe several things about this sample program. First, we see that the program uses only capital letters, since the teletypewriter has only capital letters.

A second observation is that each line of the program begins with a number. These numbers are called line numbers and serve to identify the lines, each of which is called a statement. Thus, a program is made up of statements, most of which are instructions to the computer. Line numbers also serve to specify the order in which the statements are to be performed by the computer. This means that you may type your program in any order. Before the program is run, the computer sorts out and edits the program, putting the statements into the order specified by their line numbers. (This editing process facilitates the correcting and changing of programs, as we shall explain later.)

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

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

With this preface, let us go through the example, step by step. The first statement, 10, is a READ statement. It must be accompanied by one or more DATA statements. When the computer encounters a READ statement while executing your 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 the 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, we have exhausted the available data in statement 70, but there is more in statement 80, and we pick up from it the number 2 to be assigned to E.

We next go to statement 15, which is a LET statement, and first encounter a formula to be evaluated. (The asterisk "*" is obviously used to denote multiplication.) In this statement we direct the computer to compute the value of AE- BD, and to 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 equals sign. We know that 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. If the computer discovers a "yes" answer to the question, it is directed to go to line 65, where it prints "NO UNIQUE SOLUTION". From this point, it would go to the next statement. But lines 70, 80, and 85 give it no instructions, since DATA statements are not "executed", and it then goes to line 90 which tells it to "END" the program.

If the answer to the question "Is G equal to zero?" is "no", as it is in this example, the computer goes on to the next statement, in this case 30. (Thus, an IF-THEN tells the computer where to go if the "IF" condition is met, but to go on to the next statement if it is not met.) The computer is now directed to read the next two entries from the DATA statements, -7 and 5, (both are in statement 80) and to assign them to C and F respectively. The computer is now ready to solve the system

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

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

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

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

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

In line 30 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 to print out the solutions. It is directed back again to 30, but there are no more pairs of numbers available for C and F in the DATA statements. The computer then informs you that it is out of data, printing on the paper in your teletypewriter "OUT OF DATA IN 30" and stops.

For a moment, let us look at the importance of the various statements. For example, what would have happened if we had omitted line number 55? The answer is simple: the computer would have solved the three systems and then told us when it was out of data. However, since it was not asked to tell us (PRINT) its answers, it would not do it, and the solutions would be the computer's secret. What would have happened if we had left out line 20? In this problem just solved, nothing would have happened. But, if G were equal to zero, we would have set the computer the impossible task of dividing by zero in 37 and 42, and it would tell us so emphatically, printing "DIVISION BY ZERO IN 37" and "DIVISION BY ZERO IN 42." Had we left out statement 60, 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". It would do this and then stop.

One very natural question arises from the seemingly arbitrary numbering of the statements: why this selection of line numbers? The answer is that the particular choice of line numbers is arbitrary, as long as the statements are numbered in the order which we want the machine to follow in executing the program. We could have numbered the statements 1, 2, 3, . . . , 13, although we do not recommend this numbering. We would normally number the statements 10, 20, 30, . . . , 130. We put the numbers such a distance apart so that we can later insert additional statements if we find that we have forgotten them in writing the program originally. 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; and in the editing and sorting process, the computer will put them in their proper place.

Another question arises from the seemingly arbitrary placing of the elements of data in the DATA statements: why place them as they have been in the sample program? Here again, the choice is arbitrary and we need only put the numbers in the order that we want them read (the

first for A, the second for B, the third for D, the fourth for E, the fifth for C, the sixth for F, the seventh for the next C, etc.). In place of the three statements numbered 70, 80, and 85, we could have put

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

or we could have written, perhaps more naturally,

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

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

The program and the resulting run is shown below exacly as it appears on the teletypewriter:

```
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    10:37   DEC. 17, 1965

 4              -5.5
 .666667         .166667
-3.66667         3.83333

OUT OF DATA IN  30

TIME:   0 SECS.
```

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

Note that the computer, before printing out the answers, printed the name which we gave to the problem (LINEAR) and the time and date of the computation. At the end of the printed answers the machine tells us, to the nearest second, the amount of computing time used in our problem. Since it took (considerably) less than one-half of a second for the computer to solve the three systems, the time is recorded as 0 seconds.

2.2 FORMULAS

The computer can perform a great many operations--it can add, subtract, multiply, divide, extract square roots, raise a number to a power, and find the sine of a number (on an angle measured in radians), etc.--and we shall now learn how to tell the computer to perform these various operations and to perform them in the order that we want them done.

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

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

We must be careful with parentheses to make sure that we group together those things which we want together. We must also understand the order in which the computer does its work. For example, 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. This is the same convention as is usual for $A + B\ C^D$. If this is not the order intended, then we must use parentheses to indicate a different order. For example, if it is the product of B and C that we want raised to the power D, we must write A + (B * C) ↑ D; or, if we want to multiply A + B by C to the power D, we write (A + B) * C↑D. We could even add A to B, multiply 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:

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

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

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

These rules are illustrated in the previous example. The rules also 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.

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

Functions	Interpretation	
SIN (X)	Find the sine of X	X interpreted as a number, or as an angle measured in radians
COS (X)	Find the cosine of X	
TAN (X)	Find the tangent of X	
ATN (X)	Find the arctangent of X	
EXP (X)	Find e^X	
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 ($\sqrt{X}$)	

Two other mathematical functions are also available in BASIC: INT and RND; these are reserved for explanation in Chapter 3. 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, sitting at the teletypewriter, you need the value of $(\frac{5}{6})^{17}$, you can write the two-line program.

```
10  PRINT (5/6)↑17
20  END
```

and the computer will find the decimal form of this number and print it out in less time than it took you to type the program.

Since we have mentioned numbers and variables, we should be sure that we understand how to write numbers for the computer and what variables are allowed.

2.2.1 Numbers

A number may be positive or negative and it may contain up to nine digits, but it must be expressed in decimal form. For example, all of the following are numbers in BASIC: 2, -3.675, 123456789, -.987654321, and 483.4156. The following are not numbers in BASIC: 14/3, $\sqrt{7}$, and .00123456789. The first two are formulas, but not numbers, and the last one has more than nine digits. We may ask the computer to find the decimal expansion of 14/3 or $\sqrt{7}$, and to do something with the resulting number, 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 .00123456789 in a form acceptable to the computer in any of several forms: .123456789E-2 or 123456789E-11 or 1234.56789E-6. We may write ten million as 1E7 and 1965 as 1.965E3. We do not write E7 as a number, but must write 1E7 to indicate that it is 1 that is multiplied by 10^7.

2.2.2 Variables

A variable in BASIC is denoted by any letter, or by any letter followed by a single digit. 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. and INPUT

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

Six other mathematical symbols are provided for in BASIC, symbols of relation, and these are used in IF-THEN statements where it is necessary to compare values. An example of the use of these relation symbols was given in the sample program in section 1. Any of the following six standard relations may be used:

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

2.3 LOOPS

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

The programs which use loops can, perhaps, be best illustrated and explained by two programs for the simple task of printing out a table of the first 100 positive integers 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 the line 20 is printed both 1 and its square root. 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 $4 \leq 100$ go back to line 20), etc.-- until the loop has been traversed 100 times. Then, after it has printed 100 and its square root has been printed, 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).

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

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

In line 10, X is set equal to 1, and a test is set up, like that of line 40 above. Line 30 carries out two tasks: X is increased by 1, and the test 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--and they are easier to use.

Note that the value of X is increased by 1 each time we go through the loop. If we wanted a different increase, 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 through, 11 on the third time, 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), then 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
```

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

Allowed

```
FOR X
  FOR Y
  NEXT Y
NEXT X
```

Not Allowed

```
FOR X
  FOR Y
NEXT X
  NEXT Y
```

Allowed

```
FOR X
  FOR Y
    FOR Z
    NEXT Z
  FOR W
  NEXT W
  NEXT Y
  FOR Z
  NEXT Z
NEXT X
```

2.4 LISTS and TABLES

In addition to the ordinary variables used by BASIC, there are variables which can be used to designate the elements of a list or of a table. These are used where we might ordinarily use a subscript or a double subscript, for example the coefficients of a polynomial (a_0, a_1, a_2, ...) or the elements of a matrix ($b_{1,1}$). The variables which we use in BASIC consist of a single letter, which we call the name of the list, followed by the subscripts in parentheses. Thus, we might write A(0), A(1), A(2), etc. for the coefficients of the polynomial and B(1, 1), B(1,2), etc. for the elements of the matrix. NEGATIVE subscripts are not accepted. Subscripts must be INTEGERS.

A list or table can be entered as DATA, or it can be the result of a computer calculation.

NOT MORE THAN 2 SUBSCRIPTS (IN BASIC)

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, 7, 2.2, 4, -9, 123, 4, -4, 3
```

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 form as typed would allow for the lengthening of the list by changing only statement 60, so long as it did 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 dimension 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).

On the next page is a list and run of a problem which uses both a list and a table. The program computes the total sales of each of five salesmen, all of whom sell the same three products. The list P gives the price/item of the three products and the table S tells how many items of each product which each man sold. You can see from the program the product number 1 sells for $1.25 per item, number 2 for $4.30 per item, and number 3 for $2.50 per item; and also that salesman number 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 10, 20, 30, using data in line 900, and the sales table in lines 40-80, using data in lines 910-930. The same program could be used again, modifying only line 900 if the prices change, and only lines 910-930 to enter the sales in another month.

This sample program did not need a dimension 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.

```
SALES1     10:49   20 DEC. 1965

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

RUN


SALES1     10:50   20 DEC. 1965

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


TIME:    0 SECS.
```

2.5 USE OF THE TIME-SHARING SYSTEM

Now that we know something about writing a program in BASIC, how do we set about using a teletypewriter to type in our program and then to have the computer solve our problem?

There are more details of the Time-Sharing System in Appendix C, but we shall learn enough in this section to handle a simple problem.

Sitting down at the teletypewriter, you first push the button labeled ORIG. This turns on the teletypewriter. You wait for the dial tone and then dial the computer number. When the computer answers with a "BEEP" tone you type HELLO and push the key marked RETURN. (You must, in fact, push the RETURN key after typing any line--only then does your line enter the computer.)

The computer will then type USER NUMBER--on the next line. You are to type in your user number. Again, press the RETURN key.

The computer will type SYSTEM--and you should type BASIC before hitting the return key next.

The computer then types NEW OR OLD--and you type the appropriate adjective: NEW if you are about to type a new problem and OLD if you want to recover a problem on which you have been working earlier and have stored in the computer's memory.

The computer then asks NEW PROBLEM NAME--(or OLD PROBLEM NAME, as the case may be) and you type any combination of letters, characters, and digits you like, but no more than six. In the sample problem preceding you will remember that we named it SALES 1. If you are recalling an old problem from the computer's memory, you must use exactly the same name as that which you gave the problem before you asked the computer to save it.

The computer then types READY and you should begin to type your program. Make sure that each line begins with a line number which contains no more than five digits and contains no spaces or non-digit characters. Also be sure to start at the very beginning of a line and to press the RETURN key at the completion of each line.

If, in the process of typing a statement, you make a typing error and notice it immediately, you can correct it by pressing the backward arrow (shift key above the letter "oh"). This will delete that which is in the preceding space, and you can then type in the correct character. Pressing this key a number of times will erase from this line the characters in that number of preceding spaces. The control key (to the left of the letter A) depressed with the X key will delete the entire line being typed.

After typing your complete program, you type RUN, press the RETURN key, and hope. The computer will type the name of your program, the time of day, and the date, and then analyze your program. If the program is one which the computer can run, it will then run it and type out any results for which you have asked in your PRINT statements. This does not mean that your program is correct, but that it has no errors of the type known as "grammatical errors." If it has errors of this type, the computer will type an error message (or several error messages) to you. A list of the error messages is contained in Appendix A, together with the interpretation of each.

If you are given an error message, informing you of an error in line 60, for example, you can correct this by typing a new line 60 with the correct statement. If you want to eliminate the statement on line 110 from your program, you can do this by typing 110 and then the RETURN key. If you want to insert a statement between those on lines 60 and 70, you can do this by giving it a line number between 60 and 70.

If it is obvious to you that you are getting the wrong answers to your problem, even while the computer is running, you can type STOP and the computation will cease. (If the teletypewriter is actually typing, there is an express stop--just press the "S" key.) It will then type READY and you can start to make your corrections. This action will sometimes cause the terminal to be disconnected. To avoid this, the CTRL and @ should be used to stop a run.

After you have all of the information you want, and are ready to leave the teletypewriter, you should type GOODBYE (or even BYE). The computer then types the time, and moves up your paper for ease in tearing off.

A sample use of the time-sharing system is shown below. The message "WAIT" was typed by the computer; it indicates that someone else was being served at the moment. The delay is usually no more than 10 seconds.

```
HELLO
ON AT  16:47    PX MON 08/29/66

USER NUMBER--123456
SYSTEM--BASIC
NEW OR OLD--NEW
OLD PROBLEM NAME--SAMPLE

10 FOR N = 1 TO 7
20 PRINT N, SQR (N)
30 NEXT N
40 PRINT "DONE"
50 END
RUN
WAIT.

SAMPLE     16:46    PX MON 08/29/66

 1              1
 2              1.41421
 3              1.73205
 4              2
 5              2.23607
 6              2.44949
 7              2.64575
DONE

TIME:    0 SECS.
```

2.6 ERRORS and "DEBUGGING"

It may occasionally happen that the first run of a new problem will be free of errors and give the correct answers. But it is much more common that errors will be present and will have to be

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

These corrections can be made at any time--whenever you notice them--either before or after a run. Since the computer 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.

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. If you have studied trigonometry, you know that $\pi/2$ is the correct value; but 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 <u>it</u> M. This number will be checked against SIN (.3), and so on down the line. 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 of the 31 sines, and X0 will be the argument that produced that 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 sine is the largest. Lastly, it will check the 3001 numbers 0, .001, .002, .003, ... , 2.998, 2.999, and 3, to find 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 teletypewriter, we write a program and let us assume that it is 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 teletypewriter and make explanatory comments on the right side.

```
NEW OR OLD--NEW
NEW PROBLEM NAME--MAXSIN
READY.

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


MAXSIN     11:15    DEC.17,1965

ILLEGAL FORMULA        IN 70
NEXT WITHOUT FOR       IN 80
FOR WITHOUT NEXT

TIME:      0 SECS.

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


MAXSIN     11:16    DEC.17,1965

 .1              .1              .1
 .2              .2              .1
 .3

STOP.
READY.
```

Notice the use of the backwards arrow to erase a character in line 40, which should have started IF SIN (X) etc.

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, where we see that we mixed variables. This is corrected by changing line 80.

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 STOP, even while it is running. We ponder the program for a while, trying to figure out what is wrong with it. We notice that SIN (0) is compared with M on the first time through the loop, but we had assigned no value to M. So we wonder if giving a value less than the maximum value of the sine will do it, say -1.

```
20  LET M= -1
RUN


MAXSIN     11:17   DEC.17,1965

 0              0              .1
 .1             .1             .1
 .2             .2             .1
 .3             .3             .1
 .4

STOP.
READY.

70
85  PRINT XO, M, D
RUN


MAXSIN     11:18   DEC.17,1965

 1.6            .999574        .1
 1.6            .999574        .1
 1.6            .999574        .1
 1.V

STOP.
READY.

90  GO TO 10
5   PRINT "X VALUE", "SIN", RESOLUTION"
RUN


MAXSIN     11:19   DEC.17,1965

ILLEGAL FORMULA       IN 5

TIME:   0 SECS.


5   PRINT "X VALUE", "SINE", "RESOLUTION"
RUN
```

We see that we initialized XO instead of M in line 20, so we change line 20 to give an initial value to M.

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

We fix this by moving the PRINT statement outside the loop. Typing 70 deletes that line, and line 85 is outside of loop. We also realize that we want M printed and not X.

We see that we are performing the same operation (the case for D = .1) over and over again. So we stop it and inspect the program again.

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 also decide to put in headings for our columns by a PRINT statement.

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

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

```
MAXSIN    11:19   DEC.17,1965

X VALUE          SINE              RESOLUTION
 1.6              .999574           .1
 1.57             1.                .01
 1.571            1.                .001

OUT OF DATA IN  10

TIME:   13 SECS.
```

Exactly 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 the finer subdivisions.

The whole process took little more than 13 seconds of the computer's time.

```
LIST
```

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

```
MAXSIN    11:20   DEC.17,1965

5   PRINT "X VALUE", "SINE", "RESOLUTION"
10  READ D
20  LET M= -1
30  FOR X = 0 TO 3 STEP D
40  IF SIN(X) <= M THEN 80
50  LET XO=X
60  LET M = SIN(X)
80  NEXT X
85  PRINT XO, M, D
90  GO TO 10
100 DATA .1, .01, .001
110 END

SAVE

READY.
```

The program is saved for later use. This should not be done unless future use is necessary.

In solving this problem, there are two common devices which we did not use. One is the insertion of a PRINT statement when we wonder if the machine is computing what we think we asked it to compute. For example, if we wondered about M, we could have inserted 65 PRINT M, and we would have seen the values. The other device is used after several corrections have been made and you are not sure just what the program looks like at this stage--in this case type LIST, and the computer will type out the program in its current form for you to inspect.

2.7 SUMMARY OF ELEMENTARY BASIC STATEMENTS

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

2.7.1 LET

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

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

7.2 READ and DATA

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

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 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, 3.14159265

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.

.7.3 PRINT

The PRINT statement has a number of different uses and is discussed in more detail in Chapter 3. The common uses are:

a. To print out the result of some computations
b. To print out verbatim a message included in the program
c. To perform a combination of a and b
d. To skip a line

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

Examples of type A:
```
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, and e^{AB}. The computer will compute the two formulas and print them for you, as long as you have already given values to A, B, and C. It can print up to five numbers per line in this format.

Examples of type b:
```
100 PRINT "NO UNIQUE SOLUTION"
430 PRINT "X VALUE", "SINE", "RESOLUTION"
```

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

Examples of type c:
```
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.

Example of type d:
```
250 PRINT
```

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

2.7.4 GO TO

There are times in a program when you do not want all commands executed in the order that they appear in the program. An example of this occurs in the MAXSIN problem where the computer has computed XO, M, and D and printed them out in line 85. We did not want the program to go on to the END statement yet, but to go through the same process for a different value of D. So we directed the computer to go back to line 10 with a GO TO statement. Each is of the form GO TO [line number].

Example:
```
150     GO TO 75
```

2.7.5 IF--THEN

There are times that we are interested in jumping the normal sequence of commands, if a certain relationship holds. For this we use an IF--THEN statement, sometimes called a conditional GO TO statement. Such a statement occurred at line 40 of MAXSIN. Each such statement is of the form

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

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

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

IF--THEN can be used within a definite loop (FOR--NEXT) to terminate the loop before it reaches the final value of the index. The index value at the time of exit will be retained for further computations.

2.7.6 FOR and NEXT

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

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

Most commonly, the expressions will be integers and the STEP omitted. In the latter case, a step size of one is assumed. The accompanying NEXT statement is simple in form, but the variable must be precisely the same one as that following FOR in the FOR statement. Its form is NEXT variable.

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

           120  FOR X4 = (17 + COS (Z))/3 TO 3*SQR (10) STEP 1/4
           235  NEXT X4

           240  FOR X = 8 TO 3 STEP -1

           456  FOR J = -3 TO 12 STEP 2
```

Notice that the step size may be a formula (1/4), a negative number (-1), or a positive number (2). In the example with lines 120 and 235, the successive values of X4 will be .25 apart, in increasing order. In 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 once and for all upon entering the FOR statement. The control variable can be changed in the body of the loop; of course, the exit test always uses the latest value of this variable.

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

2.7.7 DIM

Whenever we want to enter a list or a table with a subscript greater than 10, we must use a DIM statement to inform the computer to save us sufficient room for the list or the table.

```
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.

2.7.8 END

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

Example: 999 END

3. ADVANCED BASIC

3.1 MORE ABOUT PRINT

The uses of the PRINT statement were described in 2.7.3, but we shall give more detail here. Although the format of answers is automatically supplied for the beginner, the PRINT statement permits a greater flexibility for the more advanced programmer who wishes a different format for his output.

COMMA :

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

SEMICOLON :

Shorter zones can be manufactured by use of the semicolon, and the zones are six spaces long for 1-digit, 2-digit, and 3-digit numbers, nine spaces long for 4-digit, 5-digit, and 6-digit numbers, and twelve spaces long for 7-digit, 8-digit, and 9-digit numbers. As with the comma, a semicolon is a signal to move to the next short print zone or, if the last such zone has just been filled, to move to the first print zone of the next line.

For example, if you were to type the program

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

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

```
20  PRINT I,
```

you would have the numbers printed in the zones, reading

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

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

```
20  PRINT I;
```

and the result would be printed

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

You should remember that a label inside quotation marks is printed just as it appears and also that the end of a PRINT signals a new line, unless a comma or semicolon is the last symbol. An empty print zone can be generated by use of empty quotation marks: " ",
Thus, the instruction

```
50  PRINT X, Y
```

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

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

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

Since the end of a PRINT statement signals a new line, you will remember that

```
250  PRINT
```

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

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

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

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

1. If a number is an integer, the decimal point is not printed. If the integer contains more than nine digits, the teletypewriter will give you the first digit, followed by (a) a decimal point, (b) the next five digits, and (c) an E followed by the appropriate integer. For example, it will take 32,437,580,259 and write it as 3.24376 E 10.

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

3. 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 six decimal number. Thus, .03456 means that the number is exactly .0345600000, while 3.45600 E -2 means that the number has been rounded to .0345600.

4. Trailing zeros after the decimal point are not printed. The following program, in which we print out the first 45 powers of 2, shows how numbers are printed. Note that the semicolon "packed" form sometimes causes the last few characters in a number to be printed on top of each other. BASIC checks to see if there are 12 or more spaces at the end of a line before printing a number there, but some numbers require 15 spaces.

```
10 FOR I = 1 TO 45
20 PRINT 2↑I;
30 NEXT I
40 END
RUN


PRINTE    15:45   PX MON 08/29/66

 2     4     8     16    32    64    128   256   512   1024    2048
 4096     8192     16384    32768    65536    131072   262144   524288
 1048576     2097152     4194304     8388608     16777216    33554432
 67108864    134217728   268435456   536870912   1.07374 E 9    2.14748E
 4.29497 E 9    8.58993 E 9    1.71799 E 10   3.43597 E 10   6.87195 E 1
 1.37439 E 11   2.74878 E 11   5.49756 E 11   1.09951 E 12   2.19902 E 1
 4.39805 E 12   8.79609 E 12   1.75922 E 13   3.51844 E 13

TIME:    0 SECS.
```

72 spaces/line available

3.2 FUNCTIONS

There are two functions which were listed in Section 2.2 but not described. These are INT and RND.

The INT function is the function which frequently appears in algebraic computation as $[x]$, and it gives the greatest integer not greater than x. Thus INT(2.35) = 2, INT (-2.35) = -3, and INT (12) = 12.

One use of the INT function is to round numbers. We may use it to round to the nearest integer by asking for INT (X + .5). This will round 2.9, for example, to 3, by finding:

INT (2.9 + .5) = INT (3.4) = 3.

You should convince yourself that this will indeed do the rounding guaranteed for it (it will round a number midway between two integers up to the larger of the integers).

It can also be used to round to any specific number of decimal places. For example, INT (10*X + .5)/10 will round X correct to one decimal place, INT (100*X + .5)/100 will round X correct to two decimal places, and INT (X*10 ↑ D + .5)/10 ↑ D round X correct to D decimal places.

The function RND produces a random number between 0 and 1. The form of RND requires an argument, although the argument has no significance, and so we write RND(X) or RND (Z).

If we want the first twenty random numbers, we write the program below and we get twenty six-digit decimals. This is illustrated in the following program.

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


RNDTES    10:56   20 DEC. 1965

 .746489          .196691          5.33676 E-2      .32369           .244322
 .625169          .19313           .935845          .445447          .26231
 .218802          .783032          .4026            .84835           .558119
 .980484          .918514          .873523          .388814          .393435

TIME:     0 SECS.
```

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

```
20 PRINT INT (10*RND(X)),
```

and we would then obtain

```
RNDTES    10:58   20 DEC. 1965

 7                1                0                3                2
 6                1                9                4                2
 2                7                4                8                5
 9                9                8                3                3

TIME:    0 SECS.
```

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

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

RNDTES    11:00   20 DEC. 1965

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

TIME:    0 SECS.
```

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

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


RNDTES     11:01    20 DEC. 1965

 19     8      6      11     9      17     8      23     13     10     9
 20     13     21     16     24     23     22     12     12

TIME:    0 SECS.
```

In general, if we want our random numbers to be chosen from the A integers of which B is the smallest, we would call for

INT (A*RND(X) + B).

If you were to run the first program of this section again, you would get the same twenty numbers in the same order. But we can get a different set by "throwing away" a certain number of the random numbers. For example, in the following program we find the first ten random numbers and do nothing with them. We then find the next twenty and print them. You will see, by comparing this with the first program, that the first ten of these random numbers are the second ten of the earlier program.

```
10 FOR I = 1 TO 10
20 LET Y = RND (X)
30 NEXT I
40 FOR I = 1 TO 20
50 PRINT RND(X),
60 NEXT I
70 END
RUN


RNDTES     11:03    20 DEC. 1965

 .218802          .783032          .4026            .84835           .558119
 .980484          .918514          .873523          .388814          .393435
 .545924          .578063          .638623          .637121          .587565
 .952204          .985279          7.67761 E-2      9.61704 E-2      .736181

TIME:    0 SECS.
```

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

The handiness of such a function can be seen in a program where you frequently need the function e^{-x^2}. You would introduce the function by the line

```
30 DEF FNE (X) = EXP(-X ↑ 2)
```

and later on call for various values of the function by FNE(.1), FNE(3.45), FNE(A+2), etc. Such a 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 fit 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 the one denoting the argument of the function. Thus, assuming FNR is defined by

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

if you have previously assigned values to Y and Z, you can ask for FNR (2.175). You can give new values to Y and Z before the next use of FNR.

The use of DEF is generally limited to those cases where the value of the function can be computed within a single BASIC statement. Often much more complicated functions, or even pieces of a program, must be calculated at several different points within the program. For these functions, the GOSUB statement may frequently be useful, and it is described in the next section.

3.3 GOSUB and RETURN

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

```
90 GOSUB 210
```

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

```
350 RETURN
```

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

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

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

The user must be very cardful not to write a program in which a GOSUB appears inside a subroutine which refers to one of the subroutines already entered. (Recursion is <u>not</u> allowed!)

```
GCN3NO    11:08   20 DEC. 1965

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


GCN3NO    11:09   20 DEC. 1965

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

OUT OF DATA IN  20


TIME:   0 SECS.
```

3.4 INPUT

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

```
40 INPUT X, Y
```

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

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

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

and the machine will type out

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

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

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

3.5 SOME MISCELLANEOUS STATEMENTS

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

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

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

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

Sometimes it is necessary to use the data in a program more than once. The RESTORE statement permits reading the data as many additional times as it is used. Whenever RESTORE is

encountered in a program, the computer restores the data block pointer to the first number. A subsequent READ statement will then start reading the data all over again. A word of warning--if the desired data are preceded by code numbers or parameters, superfluous READ statements should be used to pass over these numbers. As an example, the following program portion reads the data, restores the data block to its original state, and reads the data again. Note the use of line 570 to "pass over" the value of N, which is already known.

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

3.6 MATRICES

Although you can work out for yourself programs which involve matrix computations, there is a special set of eleven instructions for such computations. They are identified by the fact that each instruction must start with the word 'MAT'. They are

MAT READ A, B, C	Read the three matrices, their dimensions having been previously specified.
MAT PRINT A, B; C	Print the three matrices, with A and C in the regular format, but B closely packed.
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 matrix B.
MAT C = INV (A)	Invert the matrix A.
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 a formula.
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.

Special rules apply to the dimensioning of matrices which occur in MAT instructions. To begin with, each such matrix must be declared in a DIM statement (not a MAT DIM statement, just a DIM statement). 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. Then, before any computation is carried out, the precise dimensions must be specified. This may be accomplished by any one of four MAT instructions:

```
MAT READ C(M,N)
MAT C = ZER (M,N)
MAT C = CON(M,N)
MAT C = IDN (N,N).
```

Since each matrix has a column numbered 0 and a row numbered 0, the first three instructions would specify matrices of size (M+1) x(N+1) and the last, since an identity matrix must be square, a matrix (N+1) x (N+1). These same instructions may also be used to change the dimension of a matrix, as long as it does not exceed the dimensions declared in the DIM statement.

Since each matrix has a column numbered 0 and a row numbered 0, the instruction MAT C = CON(2,3) sets up a 3x4 matrix with rows 0, 1, and 2, and columns 0, 1, 2, and 3:

NO. ROWS - 1
NO. COLUMNS - 1

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

While the combination of ordinary BASIC instructions and MAT instructions makes the language much more powerful, the user has to be very careful about his dimensions. In addition to having both a DIM statement, and a declaration of current dimension, care must be taken with the 11 MAT instructions. For example, a matrix product MAT C = A*B may be illegal for one of two reasons: A and B may have dimensions such that the product is not defined, or even if it is defined, C may have the wrong dimensions for the answer. In either case a "DIMENSION ERROR" message results.

Vectors may be used in place of matrices, as long as the above rules are observed. Since a vector like X (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 0. Thus

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

introduces an 8-component column vector and a 6-component row-vector.

There is room for a total of about 2000 components in all vectors and matrices--less if the program is long.

The same matrix may occur on both sides of an MAT equation in case of addition, subtraction, or constant multiplication; but not in any of the other instructions. Thus

```
MAT A = A+B   MAT A = (2.5)*A   MAT A = A-A   MAT B = A*A
```

are all legal. Note that the fourth example (matrix multiplication) is also legal, but that

```
MAT A = B*A
```

will result in nonsense. At the moment there is no instruction of the form: MAT A = B, but the same goal is achieved by

```
MAT A = (1) *B
```

Also, only a single arithmetical operation is allowed; MAT A = A+B-C is illegal, but may be achieved by 2 MAT instructions.

We close with two illustrations of matrix programs. The first one reads in A and B, and uses C for answers. First A+A is computed, then A*B. Note that correct dimensions are set up in lines 30 and 40. Note also that M = 1, N = 2 results in A being 2X3 and B being 3X3. Both MAT PRINT formats are illustrated, and one method of labeling a matrix print is shown.

```
MATRIX     11:32   20 DEC. 1965

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 = ZER(M,N), B(N,N)
100 MAT C = A+A
120 MAT PRINT C;
140 MAT C = A*B
150 PRINT "A*B ="
160 PRINT
170 MAT PRINT C
190 DATA 1,2
191 DATA 1,2,3
192 DATA 4,5,6
193 DATA 1,0,-1
194 DATA 0,-1,-1
195 DATA -1,0,0
199 END

RUN


MATRIX     11:32   20 DEC. 1965

 2      4      6

 8      10     12

A*B =

-2              -2              -3

-2              -5              -9


TIME:    1 SECS.
```

The second example inverts an (N+1) x (N+1) Hilbert Matrix:

```
 1      1/2     1/3  . . . 1/(N+1)
1/2     1/3     1/4  . . . 1/(N+2)
1/3     1/4     1/5  . . . 1/(N+3)
 .       .       .   . . .    .
 .       .       .   . . .    .
 .       .       .   . . .    .
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. In this example, we have supplied 3 for N in the DATA statement and have made a run for the 4x4 case.

```
MATRIX     11:42    20 DEC. 1965

10 DIM A(20,20), B(20,20)
20 READ N
30 MAT A = CON(N,N)
40 MAT B = CON(N,N)
50 FOR I = 0 TO N
60 FOR J = 0 TO N
70 LET A(I,J) = 1/(I+J+1)
80 NEXT J
90 NEXT I
100 MAT B = INV(A)
110 MAT PRINT B;
190 DATA 3
199 END

RUN


MATRIX     11:42    20 DEC. 1965

 16.      -120.      240.     -140.

-120.      1200.    -2700.     1680.

 240.     -2700.     6480.01  -4200.

-140.      1680.    -4200.     2800.


TIME:    1 SECS.
```

It may be of interest that a 20 x 20 matrix is inverted in about 6 seconds, but the reader is warned that beyond N = 6 (the 7 x 7 case) the Hilbert matrix cannot be inverted because of severe roundoff errors.

APPENDIX A
ERROR MESSAGES

The various error messages that can occur in BASIC, together with their interpretation, are now given:

Error Message	Interpretation
DIMENSION TOO LARGE	The size of a list or table is too large for the available storage. Make them smaller. (See Appendix B.)
ILLEGAL CONSTANT	More than nine digits or incorrect form in a constant number, or a number out of bounds (> 5.78960E76).
ILLEGAL FORMULA	Perhaps the most common error message, may indicate missing parentheses, illegal variable names, missing multiply signs, illegal numbers, or many other errors. Check the statement thoroughly.
ILLEGAL RELATION	Something is wrong with the relational expression in an IF-THEN statement. Check to see if you used one of the six permissable relational symbols.
ILLEGAL LINE NUMBER	Line number is of incorrect form, or contains more than five digits.
ILLEGAL INSTRUCTION	Other than one of the sixteen legal BASIC instructions has been used following the line number.
ILLEGAL VARIABLE	An illegal variable name has been used.
INCORRECT FORMAT	The format of an instruction is wrong. See especially IF-THEN's and FOR's.
END IS NOT LAST	Self-explanatory, it also occurs if there are two or more END statements in the program.
NO END INSTRUCTION	The program has no END statement.
NO DATA	There is at least one READ statement in the program, but no DATA statements.
UNDEFINED FUNCTION	A function such as FNF ( ) has been used without appearing in a DEF statement. Check for typographical errors.
UNDEFINED NUMBER	The statement number appearing in a GOTO or IF-THEN statement does not appear as a line number in the program.
PROGRAM TOO LONG	Either the program itself is too long for the available storage, or there are too many constants. (See Appendix B.)

TOO MUCH DATA	There is too much data in the program. (See Appendix B.)
TOO MANY LOOPS	There are too many FOR-NEXT combinations in the program. The upper limit is 26. (See Appendix B.)
NOT MATCH WITH FOR	An incorrect NEXT statement, perhaps with a wrong variable given. Also, check for incorrectly nested FOR statements.
FOR WITHOUT NEXT	A missing NEXT statement. This message can also occur in conjunction with the previous one.
CUT PROGRAM OR DIMS.	Either the program is too long, or the amount of space reserved by the DIM statements is too much, or a combination of these. This message can be eliminated by either cutting the length of the program, or by reducing the size of the lists and tables, reducing the length of printed labels, or reducing the number of simple variables.

The following error messages can occur after your program has run for awhile. Thus, they may conceivably occur after the first part of your answers have been printed. All of these errors indicate the line number in which the error occurred.

OUT OF DATA	A READ statement for which there is no DATA has been encountered. This may mean a normal end of your program, and should be ignored in those cases. Otherwise, it means that you haven't supplied enough DATA. In either case, the program stops.
SUBSCRIPT ERROR	A subscript has been called for that lies outside the range specified in the DIM statement, or if no DIM statement applies, outside the range 0 through 10. The program stops.
RETURN BEFORE GOSUB	Occurs if a RETURN is encountered before the first GOSUB during the running of a program. (Note: BASIC does not require the GOSUB to have an earlier statement number--only to perform a GOSUB before performing a RETURN.) The program stops.
GOSUB NESTED TOO DEEPLY	Too many GOSUBS without a RETURN. It may mean that subroutines are being exited by GOTO or IF-THEN statements rather than by RETURNs. The program stops.
DIVISION BY ZERO	A division by zero has been attempted. The computer assumes the answer is + ∞ (about 5.78960E76) and continues running the program.
DIMENSION ERROR	A dimension inconsistency has occurred in connection with one of the MAT statements. The program stops.
NEARLY SINGULAR MATRIX	The INV operation in MAT has encountered a matrix with zero or nearly zero pivotal elements. The matrix being inverted is singular or nearly so. The user is warned, however, that this error check is not 100 percent reliable. For instance, this error message need not occur even if the inverse is meaningless, as with high order Hilbert matrices. If this error occurs, the program stops.

ZERO TO A NEGATIVE POWER	A computation of the form $0 \uparrow (-1)$ has been attempted. The computer supplies $+\infty$ (about 5.78960E76) and continues running the program.
ABSOLUTE VALUE RAISED TO POWER	A computation of the form $(-3) \uparrow 2.7$ has been attempted. The computer supplies $(ABS(-3)) \uparrow 2.7$ and continues. Note: $(-3) \uparrow 3$ is correctly computed to give -27.
OVERFLOW	A number larger than about 5.78960E76 has been generated. The computer supplies + (or -) ∞ (about $\pm$ 5.78960E76) and continues running the program.
UNDERFLOW	A number in absolute size smaller than about 4.31809E-78 has been generated. The computer supplies 0 and continues running the program. In many circumstances, underflow is permissable and may be ignored.
EXP TOO LARGE	The argument of the exponential function is $\geq$ 176.753. $+\infty$ (5.78960E76) is supplied for the value of the exponential, and the running is continued.
LOG OF NEGATIVE NUMBER	The program has attempted to calculate the logarithm of a negative number. The computer supplies the logarithm of the absolute value and continues.
LOG OF ZERO	The program has attempted to calculate the logarithm of 0. The computer supplies $-\infty$ (about -5.78960E76) and continues running the program.
SQUARE ROOT OF A NEGATIVE NUMBER	The program has attempted to extract the square root of a negative number. The computer supplies the square root of the absolute value and continues running the program.

APPENDIX B
LIMITATIONS ON BASIC

There are some limitations imposed on BASIC by the limited amount of computer storage. Listed below are some of these limitations, in particular, those that are related to the error messages in Appendix A. The reader should realize that while the BASIC language itself is fixed, in time some of these limitations may be relaxed slightly.

Item	Limitation
Length of program	Difficult to relate to the BASIC program, but in general about two feet of teletypewriter paper filled with BASIC statements is about it.
Constants	The total number of different constants must not exceed 75.
Data	There can be no more than 1280 data numbers.
FOR statements	There can be no more than 26 FOR statements in a program.
GO TO and IF-THEN statements	The total number of these statements combined cannot exceed 80.
Lists and Tables	The total number of elements in all the lists and tables combined cannot exceed something less than 2000.

APPENDIX C
USING THE TIME-SHARING SYSTEM

The Time-Sharing System consists of a GE-235 computer with a number of input-output stations (currently, models 33 and 35 teletypewriter machines). Individuals using the input-output stations are able to "share" the use of the computer with each other in such a way as to suggest that each has sole use of the computer. The teletypewriters are the devices through which the user communicates with the computer.

THE KEYBOARD

The teletypewriter keyboard is a standard typewriter keyboard for the most part. There are 3 special keys that the user must be familiar with.

RETURN	This key is located at the right-hand end of the third row of keys, and does more than act as an ordinary carriage return. The computer ignores the line being typed until this key is pushed.
Control plus X	The Control key is located at the left-hand end of the third row of keys. When it is depressed in conjunction with the X key, the computer deletes the entire line being typed. This also acts as a carriage return.
	This key is located on the "oh" key when either SHIFT key is pressed. It is used to delete the character or space immediately preceding the "←". If this key is pressed N times, the characters or spaces in the N preceding spaces will be deleted.

ABCWT←← DE appears as ABCDE when RETURN is pushed.
AB C←←← CDE appears as ACDE when RETURN is pushed.

(Some languages available on the time-sharing system use the three characters "\", "[", and "]." They are located on the keys "L", "K", and "M" respectively when either SHIFT key is pushed.)

TELETYPEWRITER OPERATION . . . .

Besides the keyboard itself there are 4 buttons necessary to operate the machine.

BUTTON	LOCATION	FUNCTION
ORIG	leftmost of six small buttons on the right.	turns on the teletypewriter and connects it to the phone line.
CLR	next to ORIG	turns off teletypewriter and disconnects the phone circuit.

BUTTON	LOCATION	FUNCTION
LOC LF	left of the space bar on model 35 teletypewriters only.	feeds paper to permit tearing off.
BUZ-RLS	rightmost of six small buttons on the right.	turns off buzzer, which signals low paper supply.

If the teletypewriter is on a direct line to the computer, pushing the ORIG button is all that is necessary to connect up with the computer. To disconnect from the computer, type GOODBYE or BYE. If that fails, push CLR.

In order to connect with the computer from a teletypewriter, follow this routine:

1. Push the ORIG button and wait for dial tone.
2. Dial one of the dataphones at the Time-Sharing Center.

In order to disconnect from a long distance teletypewriter, type GOODBYE or BYE. If that fails, push CLR.

REQUIRED STATEMENTS AT SIGN-ON

~~Once the teletypewriter is connected to the computer, you must type HELLO.~~ Remember that all typed lines must be followed by a carriage return (RETURN). The machine will then ask for certain information which you will supply by typing the information when asked for it, and following each response with a carriage return, (R).

First, it asks for the user's number, which is assigned by the Time-Sharing Center. Next it asks for the system to be used (BASIC, ALGOL, etc.). Then it will ask whether it is a new or old program you will be working on. A new program is one which the user is about to start on, while an old program has been saved in memory for future use.

Finally, it will ask for the new or old problem name. After the machine types READY the user may begin with his new program or pick up where he left off on his old program. A typical HELLO sequence follows. (The underline indicates information typed by the user.)

HELLO ← Use to change to another USER NO.

USER NUMBER--999999 (R)

SYSTEM--BASIC (R)

NEW OR OLD--NEW (R)

NEW PROBLEM NAME--M36-2 (R)

READY.

CONTROL COMMANDS

There are a number of commands that may be given to the computer by typing the command at the start of a new line (no line number) and following the command with a carriage return (RETURN).

COMMAND	MEANING
CATALOG	The computer types a list of the names of all programs currently being saved by that user.
EDIT	Gives a brief explanation of the format used in the EDIT commands.
LENGTH	Gives the user some idea of the length of the program to the nearest 200 characters. A maximum length of 6400 characters is permitted in any one program.
LIST	Causes an up-to-date listing of the program to be typed out.
LIST--xxxxx	Causes an up-to-date listing of the program to be typed out beginning at line number xxxxx and continuing to the end.
NEW	Erases the program currently being worked on and asks for a NEW PROBLEM NAME.
OLD	Erases the program currently being worked on and asks for an OLD PROBLEM NAME.
RENAME	Permits you to change the problem name of the program currently being worked on, but does not destroy the program.
RUN	Begins the computation of a program.
RUN (typed during a computation)	Gives an indication that a program is running and how much machine time has elapsed since the run began.
SAVE	Saves the program intact for later use. (To retrieve saved programs, type OLD).
SCRATCH	Destroys the problem currently being worked on, but leaves the user number and problem name intact. It gives the user a "clean sheet" to work on.
STATUS	Gives an indication of the status of the teletypewriter you are using (running, idle, or disconnected).
STOP	Stops the computation at once. It can be typed even when the teletypewriter is typing at full speed, but may cause disconnection. Use of CTRL + @ is preferred.
SYSTEM	Permits the user to change systems (BASIC, ALGOL, etc.) without going through the HELLO sequence again.
TTY	Supplies the following information: teletypewriter number, user number, language being used, program being used, and status of teletypewriter.

COMMAND	MEANING
UNSAVE	Erases a saved program from memory. The memory of the computer is finite and this command should be used to free space in memory for programs of other users.

NATIONWIDE INFORMATION PROCESSING CENTERS

CHICAGO
110 North Wacker Drive
Chicago, Illinois
Ph: 312-663-3847

CLEVELAND
1000 Lakeside Avenue
Cleveland, Ohio
Ph: 216-523-6250

DALLAS
8100 Carpenter Freeway
Dallas, Texas
Ph: 214-631-0910

DETROIT
22150 Greenfield
Detroit, Michigan
Ph: 313-398-9000

LOS ANGELES
6151 West Century Blvd.
Los Angeles, California
Ph: 213-670-8441

LYNCHBURG
2010 Atherholt Road
Lynchburg, Virginia
Ph: 703-272-2711

NEW YORK
570 Lexington Avenue
New York, New York
Ph: 212-751-1311

NORTH TEXAS
P. O. Box 540
Wichita Falls, Texas
Ph: 817-322-7861

PHOENIX
2725 North Central Avenue
Phoenix, Arizona
Ph: 602-941-3165

SAN FRANCISCO
Ninth and MacDonald Street
Richmond, California
Ph: 415-233-7924

SCHENECTADY
60 Washington Avenue
Schenectady, New York
Ph: 518-374-2211 Ext. 53049

WASHINGTON
7800 Wisconsin Avenue
Bethesda, Maryland
Ph: 301-654-9360

Progress Is Our Most Important Product

GENERAL ELECTRIC

INFORMATION SYSTEMS DIVISION

LITHO U.S.A.