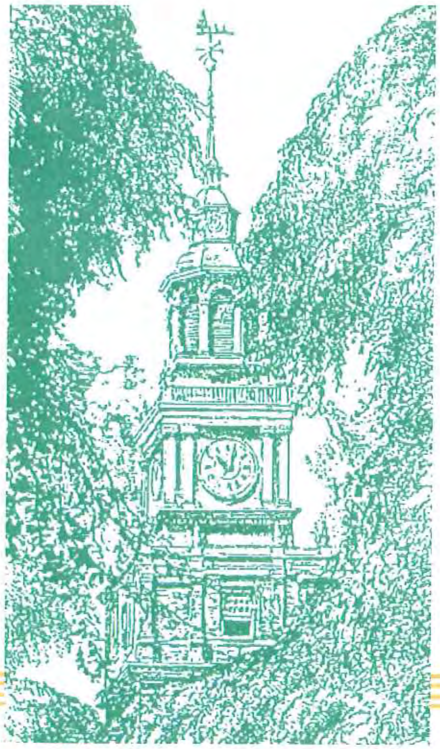
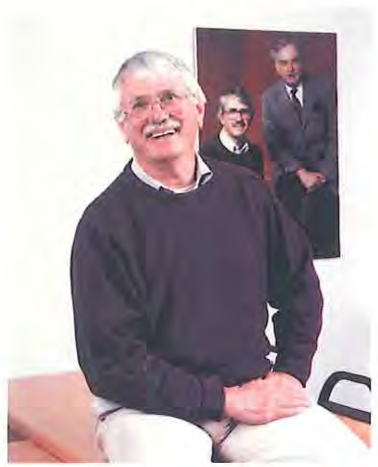


COMMEMORATIVE EDITION REPRINT

FIRST BASIC INSTRUCTION MANUAL

First Issued in May 1964



Recently, when going through some of my early BASIC files, I came across this first edition of our original Instruction Manual which was used at Dartmouth College in 1964.

I was stuck by the simplicity of our first efforts, but also, just how much of the original implementation is still used as it was those very first months of BASIC's existence.

I hope you enjoy this facsimile of that early document.

Thomas E. Kurtz – August 2000

Dartmouth College Computation Center

BASIC

Beginners' All-purpose Symbolic Instruction Code

Instruction Manual
May, 1964
First Draft

PREFACE

When plans were made for the Dartmouth College time-sharing system, which will enable 20 or more people to use the computer at the same time, the need arose for a language to meet several requirements:

1./ It should be very easy to learn. This will enable faculty and students to obtain useful information from the computer without an undue investment in learning machine languages.

2./ It should be possible to change programs from this language to the language of the machine ("compile") quickly. This is a necessity when twenty people share the time of the computer.

3./ It should be a stepping-stone for students who may later wish to learn one of the standard languages, such as FORTRAN or ALGOL.

4./ It should be a general purpose language; that is, every kind of machine computation should be programmable in it.

BASIC was constructed to meet these needs. And it has endeavored to stay as close to ordinary English as possible. As evidence for this we present, without any explanation, a program written entirely in BASIC:

```
LET X = (7+8)/3
PRINT X
END
```


TABLE OF CONTENTS

PART I

For beginners.

The instruction format.

An instruction in BASIC consists of three parts, an instruction number, an operation, and an operand. The instruction number may be any integer from 1 to 99999, chosen by the programmer. It is used purely for convenient reference. Numbers should originally be chosen with wide gaps (e.g. multiples of 10), so that further instructions may be inserted if the need arises.

The last instruction in any BASIC program is 'END'. There are only 13 other operations. Nine of these will be discussed on the following pages. Four more operations, available for the more experienced programmer, are discussed in Part II.

The operand varies in form, depending on the operation. These will be discussed as we discuss the roles of the various operations. For the convenience of the programmer, BASIC programming-forms are available. The sample program shown on page 1, written in full, on a programming sheet, would appear as follows:

Instr. no.	Operation	Operand
10	LET	X = (7+8)/3
20	PRINT	X
30	END	

Formulas.

A formula in mathematics is made up of numbers, variables, operations, and functions. The same is true in BASIC.

A number may contain up to 9 digits (the limit of accuracy of the computer), with or without a decimal point, and possibly with a minus sign. Thus the following are numbers acceptable in BASIC:

5   2.35   123456789   .123456789   -12.456

To extend the range of numbers, a factor of a power of ten may be attached, using the shorthand 'E' for ''exponent of ten''. Thus

5 E3	stands for	5×10^3 or 5000
2.39 E 52	" "	2.39×10^{52}
1.345 E-12	" "	1.345×10^{-12}

A variable in BASIC is any single letter, or a letter followed by one digit. E.g.: A, X, N5, XO, K9.

There are also variables for vectors and matrices, but these will be discussed only in Part II.

There are five arithmetical operations available in BASIC. The symbols +, -, *, / stand for addition, subtraction, multiplication, and division. Raising a quantity to a power is indicated by an upwards arrow. Since on a teletype typewriter it is impossible to print superscripts, instead of X^2 we write X↑2. However, if we want to compute X^3 and X may be negative, we must use X*X*X. The reason is that A↑B is actually interpreted as $|A|^B$.

There are 10 special functions available in BASIC:

SIN(X), COS(X), AND TAN(X) are the usual trigonometric functions, with arguments measured in radians.

ATN(X) is the inverse tangent function.

EXP(A) stands for e^A.

LOG(A) is the natural logarithm.

SQR(A) stands for $\sqrt{A}$, where the square-root is always taken of the absolute value of A.

ABS(X) stands for absolute value

RND and INT are more specialized functions. See Part II for these.

Additional functions may be introduced by definition (see p.13).

Observing the few notational conventions indicated above, mathematical formulas can be written in almost the usual manner. For example:

$x^3 + 5x^2 - 2$	would be written as	X*X*X + 5*X↑2-2
$e^{\sin(x/2)}$	"	EXP(SIN(x/2))
Arctan (cos (x.y))	"	ATN(COS(X*Y))
$\ln(\lvert x^{1/3} - y^{1/3}\rvert)$	"	LOG(ABS(X↑(1/3)-Y↑(1/3)))

LET and PRINT.

'LET' is used to compute the value of a formula and, 'PRINT' is a command to type out the answer. Their use is illustrated in the short program given previously.

The general form of a LET instruction is:

```
LET     V = F.
```

Where V may be any variable, and F any formula. For example,

```
LET     X = 2.39
LET     X = (Y + Z2)*EXP(-3)
LET    A8 = A6 + A7
LET     X = X +2
```

The last example requires an explanation: In general, the LET instruction computes the value of the formula F, and calles the answer V. Thus in the last example the present value of X is taken, 2 is added, and this is the new value of X. In other words, it simply increases X by 2.

The commonest use of PRINT is to type the value of one or more variables on a line. For example, the following pair of instructions

```
PRINT    X
PRINT    A1, A2, A3
```

Will result in having the value of X typed on the first line, and the values of A1 and A2 and A3 on the next line. As an illustration, consider the following short program:

```
10    LET      X = -.123
20    LET      Y = SIN(X)
30    PRINT    X,Y
40    END
```

The PRINT instruction can be used not only to type the value of a variable, but of a formula. Thus the above program can be shortened to read:

```
10    PRINT    -.123,    SIN(-.123)
20    END
```


Trigonometric functions have arguments measured in radians. Thus, to compute the sine of 30°, 60° and 90°, we must ask for 1/6, 1/3 and 1/2 times π radians. This can be achieved as follows:

```
10    LET    P = 3.14159265
20    PRINT  SIN(P/6), SIN(P/3), SIN(P/2)
30    END
```

We may also use the PRINT instruction to type a label, for future reference. Labels are put in quotes. Thus we could place ahead of the last program:

```
5     PRINT "SINE OF 30, 60, 90 DEGREES."
```

The program consisting of instructions 5, 10, 20, and 30 will result in the following typed answers:

```
SINE OF 30, 60, 90 DEGREES.
.5              .866025            1.
```

If there are several lines of answers to be typed, it may be convenient to leave some blank lines between typed answers. The instruction PRINT, with no operand, will skip a line.

FOR and NEXT.

The real power of computers is that on a single command of the programmer the computer will execute hundreds, thousands, or even millions of computations. The key to achieving this goal is the loop, in which the computer goes through the same set of instructions a specified number of times. Let us suppose that we wish to compute the sum $1^5 + 2^5 + 3^5 + \ldots + 100^5$. We might describe the computational method as follows, start with 0. Then add to it 1^5. Then add 2^5. and so on, until you have added 100^5. The BASIC program will look as follows:

```
10    PRINT "Sum of 100 fifth powers."
20    LET   S = 0
30    FOR   N = 1  TO  100
40    LET   S = S + N↑5
50    NEXT  N
60    PRINT S
99    END
```

The loop consists of instructions 30, 40, and 50. At first N is set equal to 1. Then 40 is carried out, adding 1^5 to our sum. Then N is set equal to 2, and 40 is again carried out, now adding 2^5 to the sum. This will be repeated 100 times. When N has been set equal to 100, and 100^5 has been added to S, the loop is complete, and the computer proceeds to instruction 60, printing the sum.

The loop is ideally designed for printing a numerical table. If one desired a table of values of e^X for X = 0, 1, ..., 20, we could write:

```
10    PRINT    "X",     "EXP(X)"
20    FOR      X = 0  TO  20
30    PRINT    X, EXP(X)
40    NEXT     X
50    END
```

The first line will then contain the labels 'X' and 'EXP(X)'. Under 'X' on each line will appear the value of X, and under 'EXP(X)' the corresponding value of e^X.

The usual form of a FOR statement is

FOR $\quad V = F_1$ TO F_2.

Where V must be a single letter variable, and while F_1 and F_2 are commonly numbers, they may be formulas as will. For example, if in the program for adding up fifth powers we had written

```
30    FOR    N = 1 TO X+Y+Z
```

then we would obtain the sum of the fifth powers of numbers up to $(x+y+z)^5$. We would, of course have to specify what x, y, and z are.

We can also use the FOR statement for fractional computations. Suppose that we desired a table of natural logarithms from 1 to 2, in intervals of 1/100. We would write:

```
10    PRINT "X" ,     "LN(X)"
20    FOR    N = 0 TO 100
30    LET    X = 1 +N/100
40    PRINT  X, LOG(X)
50    NEXT   N
60    END
```

And sometimes we wish to step through the loop in larger steps. Then we simply add the size of the step to the FOR statement. For example:

```
10    PRINT    "Sum of fifth powers of even numbers
               to 100"
20    LET      S = 0
30    FOR      N = 2 TO 100 STEP 2
40    LET      S = S + N↑5
50    NEXT     N
60    PRINT    S
99    END
```

GO TO and IF THEN

The computer normally takes the instructions in order, proceeding from one instruction to the next in the order in which they are numbered, until it hits 'END'. We have already noted one exception to this rule, namely a loop. There are many other instances where a programmer desires to overrule the normal sequence of instructions. The simplest method is the GO TO operation. This operation must be followed by an instruction number. For example, if

```
100  GO TO   50
```

occurs in the program, the computer will go back to instruction number 50 instead of going on to the next instruction.

This simple device is usually not sufficient. We often need to know what has happened so far in the computation to know whether to go on or to change our procedure. A simple example is the following:

```
........................
50  IF   X < 0  THEN 200
60 (normal procedure)
........................
200 (special procedure)
........................
```

If, when we hit 50, the variable X has a positive value, we proceed normally, but if it has a negative value, we switch to the special procedure.

This instruction is very important, since by this means we allow the previous computations to influence the future course. The most general form of the IF... THEN instruction is

$$\text{IF} \quad F_1 \propto F_2 \quad \text{THEN} \quad N$$

where F_1 and F_2 may be any formulas, N is an instruction number somewhere is our program, and $\propto$ may be any one of six relations:

=	(equal)	>=	(greater or equal)
>	(greater)	<=	(less or equal)
<	(less)	<>	(less or greater, i.e. not equal)

Suppose that we wish to find, by trial and error, where the sine function has its maximum (largest value) between 0 and 3 radians. We will search in steps of 1/100 radians.

```
10    LET     X0 = 0
20    LET     M = SIN(0)
30    FOR     N = 1  TO  300
40    LET     X = N/100
50    IF      SIN(X) <= M THEN 100
60    LET     X0 = X
70    LET     M = SIN(X)
100   NEXT    N
110   PRINT   X0, M
999   END
```

We start by choosing the value X0=0, and M = SIN(0) is the largest value so far. Within the loop 30-100, we examine the succesive values of SIN(X). If the value is no larger than the biggest so far, M, then we ignore it--by going on to the next value (see instruction 50). But if it is bigger, we note the current X, and the current value of SIN(X) as the best so far. When the loop is completed, M will be the maximum value, and X0 the location where it occurred. Note: 'X0' is 'X' followed by a zero, not by the letter 'O'.

DEF

In addition to the 10 functions provided by BASIC, the programmer may introduce his own functions, by definition. The functions will be called FNA, FNB, FNC, etc. The form of the instruction is illustrated by

```
DEF     FNC(Z) = Z↑3 + 2.7*Z
```

where Z is used just for the purpose of definition, and should not be used elsewhere in the program . The right side of the definition may be any formula containing Z. Once the above definition has been introduced, FNC(X) will result in the value $|X|^3 + 2.7X$ within any formula.

The reader may have been disturbed by the fact that the only logarithms provided were natural logarithms, and that trigonometric functions had radians as arguments. But these shortcomings are easily overcome by the DEF operation. For example, if a program starts with

```
10    LET    P = 3.14159265
20    DEF    FNS(Z) = SIN(Z*P/180)
30    DEF    FNL(Z) = LOG(Z)/LOG(10)
```

then FNS is a sine function measured in degrees, and FNL is the function log-to-the-base-10. The DEF instruction may occur anywhere within the problem.

READ and DATA

A constant may be introduced by a LET instruction, as in

```
LET P =3.14159265
```

However, this is inconvenient if there are many constants, or if the values are to be changed from one problem to the next. Instead, a list of numbers may be placed into a DATA statement. This statement may occur anywhere in the program, and usually occurs just before 'END'.

A DATA statement may contain as many numbers as can fit on a type writer line. Numbers are separated by commas. If more space is needed, additional DATA statements may be used. Up to 200 numbers may be entered by DATA statements.

The DATA statement enters the numbers into the program. To use these numbers, appropriate READ instructions must occur. The operand of this statement consists of a list of variables, separated by commas. Numbers are taken one at a time from the DATA list and assigned to these variables.

For example, in the program

```
10 READ  A, B, C
20 READ  D1, D2, D3, D4
80 DATA  1, 2, 3, 4, 5, 6
90 DATA  7, 8, 9, 10
```

A will be set equal to 1, B = 2, C = 3, D1 = 4, D2 = 5, D3 = 6, and D4 = 7.

These conventions are so designed that it is easy to change some of the constants and rerun the problem. For example, we may wish to fix A and B once and for all but vary C and D. We write:

```
        10  READ  A,B
        20  READ  C,D
        ........................
        120  GO TO 20
        130  DATA 1, 2, 1, 2, 3, 3, 4, -2
        999  END
```

Then A = 1 and B = 2 throughout. On the first pass C = 1 and D = 2, and the instructions between 20 and 120 are carried out. Then we go back to 20, and let C = 3, and D = 3, and carry out the same program. Next C = 4 and D = -2. When we return to 20 once more, we find that there is no data left, and the computer stops.

For example, let us compute the value of $x^3 + x^2 - x$ for five different values of x.

```
    10 READ X
    20 LET Y = X↑3 + X↑2 -X
    30 PRINT X, Y
    40 GO TO 10
    50 DATA 1.23, -.2345, 12.3, 1E-17, 123456E15
    99 END
```


Use of the teletypes.

Teletype typewriters are like ordinary typewriters, only they are suitable for transmitting the typed messages over telephone lines. With minor differences they are operated like ordinary typewriters.

Perhaps the most notable difference is the fact that all letters appear as capitals. Hence there is no difference in a program between 't' and 'T'. A number of special symbols appear on the key board, many of which are used in typing BASIC. There are also three special-purpose keys:

"RETURN" at the end of the second row of keys is the ordinary carriage return. But it plays a crucial role in our system. The computer completely ignores a typed line until this key is pushed. It is important to remember to push it after each line of the program and after each command.

" ← " on top of the letter 'O' erases the last character typed. This is very convenient when the programmer notices immediately that he mistyped a letter or a symbol. Pushing the backwards arrow more than once will delete several characters. For example, the line

ABCWT ← ← DE

will appear to the computer as

ABCDE

"ALT MODE" at the beginning of the second line of keys is used to delete an entire typed line. It must be pushed instead of the carriage return.

When a new programmer sits down at the teletype (or when he wishes to start a new problem), he must start by typing the word HELLO.

At this command the machine will ask for certain information. First it asks for the user's number. For undergraduates this should be the 6-digit number on the college I. D. card: A special number will be assigned by the Computation Center for others. Next is asks for the problem number. This is up to the programmer, for his future reference. Any combination of letters, numbers, and symbols up to 6 characters is allowed. Next the computer asks for the name of the system to be used. For programmers using BASIC this system is, of course, "BASIC". Finally, it asks whether the program will be typed, or whether it is an old program. When first entering a program, the correct response is 'TYPE'. We show a typical sequence of responses. The underlined statements were typed by the user (with a return after each line !!):

```
HELLO
USER NO.--446425
PROBLEM NO.--M36-2
SYSTEM--BASIC
TYPE OR OLD--TYPE
READY.
```

At this point the user types his program as indicated previously in the manual.

After the last line of the program has been typed, the user types the command

RUN

and the computation begins. All answers will be preceded by typing the user's number and the name of his problem. The typing ends with an indication of how long the computation took (waiting time does not count). Since this is computed to the nearest second, the user should not be shocked by the statement

TIME: 0 SECS.

After making a number of alterations in one's program, there may be some doubt as to exactly how the present version of the program reads. A copy of the up-to-date version is obtained by typing

LIST

The greatest advantage of the time-sharing system is that most users can complete their work in one sitting. But if this does not prove feasible, the user may type

SAVE

and have his program placed in the storage area of the Disk-memory. It will then be available to him for a reasonable period (currently one month), by typing the HELLO sequence. For example, the program on the previous page, if saved, may be obtained next time by typing:

```
HELLO
USER NO.--446425
PROBLEM NO.--M36-2
TYPE OR OLD--OLD
```

There will be a waiting period till the problem is found on the disk, and the machine types 'READY.' It is then advisable to type 'LIST' before running the program.

Since the space on the disks is limited, users are asked not to save a program unless they really expect to use it again.

If anything goes wrong, the programmer should type

STOP

This command takes effect even while the teletype is typing at full speed! Thus it may be used to stop a long output that is not what the programmer wanted. Just type STOP, and push the return key. The letters will not show until after the typing has been terminated, but the command is still effective.

Errors.

While compiling a program written in BASIC, the machine checks for various ways that the rules of BASIC may have been violated. Therefore, instead of answers, the programmer may find a variety of error messages. These are illustrated in the following example of a very poorly written program:

```
10  LET X = 2
20  READ Y
30  LET Z = XY
40  IF Z > 5 THEN 50
45  PRINT "LESS THAN 5
60  PRINT "OK"
100 END
110 GOTO 20
```

This results in the following error messages:

```
ILLEGAL FORMULA      IN 30
INCORRECT FORMAT     IN 45
END IS NOT LAST
UNDEFINED NUMBER
NO DATA
```

Hopefully, these messages are self-explanatory. The reader should test himself by finding the errors in 30 and 45. "End is not last" due to statement number 110. The undefined number (statement number) is 50. And a READ instruction must always be followed by a DATA statement.

It is very simple to make corrections by means of the teletypes. Changes must observe three rules:

(1) If a statement typed has the same number as one previously used, it replaces the statement.

(2) If a statement number is immediately followed by a carriage return, the statement of that number is deleted.

(3) If a statement with a new statement number is typed, this will be inserted in its proper place in the program.

For example, the above errors may be removed by typing

```
30 LET Z = X*Y
45 PRINT  "LESS THAN 5"
50 GO TO 60
70 GO TO 20
80 DATA 2,3
110
```

A request for a LIST now results in:

```
10 LET X = 2
20 READ Y
30 LET Z = X*Y
40 IF Z > 5 THEN 50
45  PRINT "LESS THAN 5"
50 GO TO 60
60 PRINT "OK"
70 GO TO 20
80 DATA 2,3
100 END
```

This is still not a brilliant program, but it will be accepted by the machine.

With a little practice the programmer will find that frequent <u>short</u> test-runs, followed by one or two-line corrections will get the bugs out of any program. A very useful trick is the following: Insert some extra PRINT statements (particularly in loops) during debugging. This will usually show up errors. These can be removed when the program is debugged. It is important to note that we do not type "HELLO" while correcting a program. Type "HELLO" only when you wish to start a new problem.

EXAMPLES.

Program number 1:

```
10 PRINT "AVERAGE OF 100 LOGS"
20 LET S = 0
30  FOR N = 1 to 100
40 LET S = S+ FNL (N)
50 NEXT N
60 LET S = S/100
70 PRINT S
80 DEF FNL (Z) = LOG(Z)/LOG(10)
99 END
```

Result:

```
AVERAGE OF 100 LOGS
1.5797
TIME:  3 SECS.
```

Program number 2:

```
  5 PRINT "QUADRATIC EQUATION"
  6 PRINT
 10 READ A, B, C
 20 LET D = B * B -4 * A * C
 30 LET S = - B / (2 * A )
 40 LET T = SQR(D) / (2 * A)
 50 IF D = O THEN 100
 60 IF D < 0 THEN 200
 70 PRINT A, B, C, "TWO REAL ROOTS"
 80 PRINT "ROOTS =" S + T, S - T
 90 GOTO 300
100 PRINT A, B, C, "ONE ROOT"
110 PRINT "ROOT =" S
120 GOTO 300
```


Program 2 cont.

```
200 PRINT A, B, C, "COMPLEX ROOTS"
210 PRINT "REAL PART=" S, "IMAGINARY PART=" T
300 PRINT
310 GOTO 10
400 DATA 1, -2, -3, 1, -4, 4, 1, 0, 2
999 END
```

Result:

```
QUADRATIC EQUATION
1.              -2.             -3.             TWO REAL ROOTS
ROOTS = 3.                      -1.

1.              -4.             4.              ONE ROOT
ROOT  = 2.

1.              0.              2.              COMPLEX ROOTS
READ PART  = 0.                 IMAGINARY PART = 1.41421

TIME:     1 SECS.
```

Program number 3:

```
10 PRINT "ROOT OF FUNCTION."
20 PRINT "FUNCTION IS - AT  0 AND + AT 1"
30 DEF FNF(Z) = Z↑5 +Z↑3 -1
40 LET X = 0
50 LET D = .5
60 LET Y = FNF(X)
70 IF Y = 0 THEN 200
80 IF Y > 0  THEN 150
90 LET X = X + D
100 GOTO 160
150 LET X = X - D
160 IF D ≤ .0001 THEN 200
170 LET D = D/2
180 GO TO 60
200 PRINT "ROOT =" X
999 END
```

result:

```
ROOT OF FUNCTION.
FUNCTION IS - AT 0 AND + AT 1
ROOT = .837585
TIME:  2 SECS.
```

We can find the root of a different function by changing a single instruction. For example:

```
30 DEF FNF(Z) = EXP(X) -2
```

New result:

```
ROOT OF FUNCTION.
FUNCTION IS - AT 0 AND + AT 1
ROOT = .693176
TIME:  1 SECS.
```