S. G. FRASER

digital

basic
basic
basic
basic
basic
basic
basic
basic
basic
basic
basic
basic
basic

edusystem 20 user's guide

five-user

digital equipment corporation

EduSystem-20 BASIC User's Guide

Time-Sharing, Five-User

For additional copies, order No. DEC-8I-KJXA-D from
DEC, Direct Mail, Bldg. 1-1, Maynard, Massachusetts 01754
Price $2.25

First Printing, October 1970
Second Printing, April 1971

Your attention is invited to the last two pages of this manual. Your filling in and returning the Reader's Comments page will be benificial to you and DEC. Your comments will be considered for inclusion in subsequent manuals; and, in case you require assistance, a knowledgeable DEC representative will contact you. The How To Obtain Software Information page offers you a means of keeping up-to-date with DEC's software.

Supporting Documents:

Introduction to Programming, 1970

Programming Languages, 1970

These documents may be ordered from DEC, Direct Mail, Building 1-1, Maynard, Massachusetts 01754.

PREFACE

This manual contains a comprehensive description of the EduSystem-20 BASIC[1] Programming System -- a five-user BASIC system for use on a PDP-8, -8/L, or -8/I computer with from 8K to 32K words of core memory. The manual does not require prior knowledge of the BASIC language or computers (except when loading and starting the System, Chapter 9).

The first four chapters explain and show how to use the BASIC language statements when writing your BASIC programs. Chapter five explains how to correct and edit the program so that it can be run and, if desired, saved for future use.

Chapter six reveals some advanced programming techniques which may be used to enhance program preparation and execution. Chapter seven describes how to load, start, and respond to BASIC's Initial Dialogue which configures the System to the desired user configuration.

The CAMP (Computer Assisted Mathematics Program) manuals are available for those interested in a text to teach BASIC in junior and senior high schools (even elementary schools). They were written by teachers for use by students and teachers in (specifically) grades 7 through 12. Additional manuals can be obtained from Scott, Foresman and Company, Glenview, Illinois 60025.

DEC provides one set of CAMP manuals with each machine sold, at no additional cost to the customer.

[1]BASIC (Beginner's All-purpose Symbolic Instruction Code) is a trademark registered by the Trustees of Dartmouth College.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION TO BASIC PROGRAMMING

BASIC is an easy to learn, conversational computer language for scientific, business, and educational applications. It is used to solve both simple and complex mathematical problems and is directed from the user's Teletype[1]

In writing a computer program, it is necessary to use a language or vocabulary that the computer will recognize. There are many computer languages and BASIC is one of the simplest because of the small number of readily learned commands needed, its easy application in solving problems, and its practicality in an educational environment.

BASIC is similar to other programming languages in many respects and is aimed at facilitating communication between the user and the computer. The novice computer user will benefit from reading the entire manual from the beginning The user who is already familiar with a language such as FOCAL or FORTRAN should first turn to the language summary at the end of the manual.

As a BASIC user, you type in your computational procedure as a series of numbered statements, making use of common English words and familiar mathematical notation. You can solve almost any problem by spending an hour or so learning the necessary elementary commands. After becoming more experienced, you can add the more advanced techniques needed to perform more intricate manipulations and to express your problem more efficiently and concisely. Once you have entered your statements, you give a RUN command. This command initiates the execution of your program and causes the return of your results almost instantly.

1.1 ABOUT COMPUTING

As we approach a computer terminal, there is a certain way to solve a problem. It is not enough to understand the technical commands of a computer language, we must also be able to correctly and adequately express the problem to be solved. For this reason, it will be helpful to outline the process of setting up a problem for computer solution.

The first step is to define the problem to be solved in detail. Understand each fact and possibility within the problem before attempting to go any further.

[1]Teletype is a registered trademark of the Teletype Corporation.

Problems to be solved with BASIC are generally of a level which admit to fairly straightforward analysis.

In computing there is always more than one correct way of approaching a given problem. Generally, a standard mathematical method for solution can be found, or a method developed. Programs using the same method can still be written in more than one correct way.

For some complicated programs, a flowchart is useful. A flowchart is a diagram which outlines the procedure for solving the problem, step by step.

Having a diagram of the logical flow of a problem is a tremendous advantage to you when determining the mathematical techniques to be used in solving the problem, as well as when writing the BASIC program. In addition, the flowchart is often a valuable aid when checking the written program for errors.

A flowchart is a collection of boxes and lines. The boxes indicate, in a general fashion, what is to be done; the lines indicate the sequence of the boxes. The boxes have various shapes representing the type of operation to be performed in the program (input, computation, decision, etc.).

Following satisfactory completion of a flowchart, you proceed to write the program. To do this, you need to understand the various instructions and capabilities of the BASIC language. The rest of this manual is designed to teach you how to write programs in the BASIC language in a minimal amount of time.

Once the correct procedure has been coded, it is time to try it on the computer. At this point it is possible the program will not work perfectly as originally written. BASIC will locate any mistakes the programmer has made in typing his program and print appropriate error messages to help him correct them. It is important to understand that even if the program does run, the results will be correct only if the problem has been correctly analyzed and properly written to achieve the correct solution. The computer can only do what you tell it to do. If you have unknowingly told the computer to do something other than what you wanted it to do, the results will be accurate according to the information the computer processed. The computer cannot know what you really want, only what you have told it.

1.2 HOW TO USE THIS MANUAL

The most straightforward treatment of the BASIC programming language will be obtained by reading this manual from the beginning. Examples are taken directly from Teletype output so that the reader will become familiar with the BASIC's outputs and formats. Once you have mastered the principles of the BASIC language, you will most likely only need to refer to the summaries appended to the manual.

Detailed examples appear and may be run on the computer as a first exercise before attempting an original program.

The early chapters of this manual contain directions on how to write a BASIC program. The appendix on Implementation Notes is recommended for every reader. Once you have written several BASIC programs, you will find the section on Advanced BASIC helpful; reading that section too early in your programming experience may be confusing. As soon as you are ready to try running a BASIC program on EduSystem 20, turn to Chapter 7, Running a BASIC Program.

CHAPTER 2

FUNDAMENTALS OF PROGRAMMING IN BASIC

2.1 AN EXAMPLE PROGRAM AND OUTPUT

At this point, the program in Figure 2-1 may mean little to you, although the output (following the word RUN) should be fairly clear. One of the first things you notice about the program is that each line begins with a number. BASIC requires that each line be numbered with an integer from 1 to 2046. When the program is ready to be run, BASIC executes the statements in the order of their line numbers, regardless of the order in which you typed the statements. This allows for later insertion of a forgotten or new line. The programmer is, therefore, advised to leave gaps in his numbering on the first typing of a program. Numbering by fives or tens is a common practice.

The next thing we notice about the program is that each line begins with a word, a command to tell BASIC what to do with the information on that line. Once we understand the usage of these commands we are able to describe our problem to the computer.

2.2 REM STATEMENT

The REM or REMARK statement allows the programmer to insert notes to himself or anyone who will read the program later. The form is:

```
    line-number   REM   message
```

Everything following REM is ignored by the computer. In Figure 2-1, line 10 is a remark describing what the program does. It is often useful to put the name of the program and information on what the program does at the beginning for future reference. Remarks throughout the body of a long program will help later debugging by explaining the function of each section of code within the whole program.

2.3 NUMBERS

In BASIC, as in all languages, there are conventions to be learned. The most important initial concepts are (1) how do we express a number to the computer and (2) how do we create algebraic symbols.

BASIC treats all numbers as decimal numbers, which is to say that it assumes

```
10 REMARK - PROGRAM TO TAKE AVERAGE OF
15 REMARK - STUDENT GRADES AND CLASS GRADES
20 PRINT "HOW MANY STUDENTS, HOW MANY GRADES PER STUDENT";
30 INPUT A,B
40 LET I=0
50 FOR J=0 TO A-1
55 LET V=0
60 PRINT "STUDENT NUMBER =";J
75 PRINT "ENTER GRADES"
76 LET D=J
80 FOR K=D TO D+(B-1)
81 INPUT G
82 LET V=V+G
85 NEXT K
90 LET V=V/B
95 PRINT "AVERAGE GRADE =";V
96 PRINT
99 LET Q=Q+V
100 NEXT J
101 PRINT
102 PRINT
103 PRINT "CLASS AVERAGE =";Q/A
104 STOP
140 END

RUN
HOW MANY STUDENTS, HOW MANY GRADES PER STUDENT? 5,4
STUDENT NUMBER = 0
ENTER GRADES
? 78
? 86
? 88
? 74
AVERAGE GRADE = 81.5

STUDENT NUMBER = 1
ENTER GRADES
? 59
? 86
? 70
? 87
AVERAGE GRADE = 75.5

STUDENT NUMBER = 2
ENTER GRADES
? 58
? 64
? 75
? 80
AVERAGE GRADE = 69.25

STUDENT NUMBER = 3
ENTER GRADES
? 88
? 92
? 85
? 79
AVERAGE GRADE = 86

STUDENT NUMBER = 4
ENTER GRADES
? 60
? 78
? 85
? 80
AVERAGE GRADE = 75.75

CLASS AVERAGE = 77.6

READY
```

Figure 2-1 An Example BASIC Program

a decimal point after an integer, or accepts any number containing a decimal point. The advantage of treating all numbers as decimal numbers is that the programmer can use any number or symbol in any mathematical expression, knowing that the computer can combine the numbers given. (In some languages, integers must be used separately from decimal numbers.)

A third form (other than integers and real numbers) we use in expressing numbers to BASIC is called exponential form. In this form, a number is expressed as a decimal number times some power of 10. For example:

    23.4E2

2.34E3

Max Exponent ≈ 38 See p C-1

is the same as

    2340.

The E can be read as "times 10 to the --power", depending upon the positive or negative integer following E. A number can be expressed in exponential form anywhere in the program. You may input data in any form. Results of computations are printed out as decimal numbers if they are in the range .01≤N<1,000,000. Outside this range, numbers are automatically printed out in E (exponential) format. BASIC handles seven significant digits during normal operation and input/output, as seen below:

Value Typed In	Value Printed by BASIC
.01	.01
.0099	9.900000E-3
999999	999999
1000000	1.000000E+6

BASIC automatically omits printing leading and trailing zeros in integer and decimal numbers, and formats all exponential numbers in the following form:

(sign) digit . six digits E ± exponent value

For example:

```
-3.470218E+8    is equal to    -347,021,800
 7.260000E-4    is equal to    .000726
```

All letters are printed as capitals at the Teletype console. Therefore, a convention used by programmers (and which occurs on Teletype output) to distinguish zeros from the letter "O" is to slash zeros (Ø). This enables accurate input to BASIC (when you are typing a program previously written down) and ease of understanding in reading computer output (in which zeros are slashed). Notice that unlike a typewriter, the letter "L" does not produce the number "1" on the console keyboard. All numbers are on the top row of the keyboard. Notice also that in BASIC commas are not inserted into large numbers, as we are accustomed to doing (i.e., 1,742,300 is 1742300).

2.4 VARIABLES

A variable in BASIC is an algebraic symbol for a number, and is formed by a single letter or a letter followed by a digit. For example:

Acceptable Variables	Unacceptable Variables
I	2C - a digit cannot begin a variable
B3	AB - two or more letters cannot form a variable
X	

We assign values to variables by either inputting these values or indicating them in a LET statement.

2.5 LET STATEMENT

Before examining the LET statement, we should first clarify the meaning of the equal sign (=). For example , the command:

```
10 LET A = B + C
```

tells BASIC to add the values of B and C and store the result in a variable called A (the number 10 is the line number mentioned earlier).

```
20 LET D = 7.2
```

means to store the value 7.2 in the variable D.

```
30 LET D = 406
```

causes the value of D which was 7.2 (above) to be changed to 406.

The equal sign means replacement rather than equality. In the statement:

```
10 LET X = X + 1
```

we mean "add one to the current value of X and store the result back in the variable X."

Values of variables can be reassigned throughout the program as the programmer wishes. The equal sign, then, shows a replacement relationship where the expression after the equal sign is evaluated and replaces the old value (if any) of the variable indicated.

The LET statement is of the form:

```
line-number    LET    variable    =    formula
```

where a formula is either a number, another variable, or an arithmetic expression. The LET statement is the most elementary BASIC statement, used when computation is to be performed or, to put it more generally, whenever a new value is assigned to a variable.

Since the LET statement is the most frequently used BASIC statement, the word LET is optional. The statement V = Ø is treated as LET V = Ø.

2.6 ARITHMETIC OPERATIONS

Looking at the console keyboard, we can find some of the usual arithmetic symbols (+, -, and =). BASIC can perform addition, subtraction, multiplication, division, and exponentiation as well as other more complicated operations explained later. Each mathematical formula fed to BASIC must be on a single line, with a line number and an appropriate command. The five operators used in writing most formulas are:

Operator	Meaning	Example
+	Addition	A + B
-	Subtraction	A - B
*	Multiplication	A * B
/	Division	A / B
↑	Exponentiation (raise A to the Bth power)	A ↑ B

In BASIC, the mathematical formula:

$$A = 7\left(\frac{B^2 + 4}{X}\right)$$

would be written:

```
10 LET A = 7 * ((B↑2 + 4) / X)
```

How does BASIC know what operation to perform first? There are conventions built into computer languages; BASIC performs arithmetic operations with the order of evaluation indicated by the rules below.

1. Parentheses receive the top priority. Any expression within parentheses is evaluated before an unparenthesized expression.
2. In absence of parentheses, the order of priority is:
   a. Exponentiation
   b. Multiplication and Division
   c. Addition and Subtraction
3. If 1 or 2 does not clear ambiguity, the order of evaluation is from left to right as we would read the formula.

So in the example above, B↑2 is evaluated first, then (B↑2+4) and then ((B↑2+4)/X), finally 7* ((B↑2+4)/X). Keeping the conventions above in mind, A↑B↑C will be evaluated as (A↑B)↑C, likewise A/B*C is evaluated as (A/B)*C.

2.7 PARENTHESES AND SPACES

Use of parentheses allows us to change the order of priority of evaluation in rule 2 above. They also prevent any confusion or doubt on our part as to how the expression is evaluated. To make a formula easier to write as well as read, it is frequently a good idea to provide more parentheses than strictly required. For example, which is easier to read?

```
A*B↑2/7 + B/C*D↑2
(A*B↑2)/7 + (B/C)*D↑2
((A*B↑2)/7) + ((B/C)*D↑2)
(((A*(B↑2))/7) + ((B/C)*(D↑2)))
```

Each of the above formulas will be executed the same way, but which makes the most sense to the programmer reading it, or perhaps trying to make corrections later? On the other hand, which has superfluous parentheses not required for clarity?

Spaces may also be used freely to make formulas easier to read

```
    1Ø LET B= D↑2 + 1
```

instead of

```
    1ØLETB=D↑2+1
```

2.8 FUNCTIONS

BASIC performs several mathematical calculations for the programmer, eliminating the need for tables of trig functions, square roots, and logarithms. These functions have a three letter call name (the argument X can be a number, variable, formula, or another function) and are written as follows:

Functions	Meaning
SIN(X)	Sine of X (where X is expressed in radians) is returned.
COS(X)	Cosine of X (where X is expressed in radians) is returned.
TAN(X)	Tangent of X (where X is expressed in radians) is returned.
ATN(X)	Arctangent of X is returned as an angle in radians.
EXP(X)	e^X (where e = 2.718282) is returned.
LOG(X)	Natural logarithm of X, log X, is returned.
ABS(X)	Absolute value of X, $\|X\|$, is returned.
SQR(X)	Square root of X, $\sqrt{X}$,is returned.
CHR$(X)	Character Equiv on O/P

These functions are built into BASIC and can be used in any statement as part of a formula. For example:

```
    1Ø LET A = SIN(ABS(X))/2
```

will cause A to be set equal to one-half the value of the sine of the absolute value of X.

Four other functions are available and, although they are not as readily useful to the beginning programmer, they will become so as skill in designing program logic increases.

2.8.1 Sign Function, SGN(X)

The sign function returns the value +1 if the argument X is a positive value, 0 if X is 0, and -1 if X is negative. For example:

SGN(3.42) = 1, SGN(-42) = -1, and SGN(23-23) = 0

The SGN function is exercised in the following program (READ and DATA statements are explained in Section 3.1).

```
 10 REM - SGN FUNCTION EXAMPLE
 20 READ A,B
 25 PRINT "A="A, "B="B
 30 PRINT "SGN(A)="SGN(A), "SGN(B)="SGN(B)
 40 PRINT "SGN(INT(A))="SGN(INT(A))
 50 DATA -7.32, .44
 60 END

RUN
A=-7.32        B= .44
SGN(A)=-1      SGN(B)= 1
SGN(INT(A))=-1

READY
```

2.8.2 Greatest Integer Function, INT(X)

The integer function returns the value of the greatest integer not greater than the argument. For example:

```
INT(34.67) = 34
```

INT can be used to round numbers to the nearest integer by asking for INT(X+.5).

For example:

```
INT(34.67+.5) = 35
```

INT can also be used to round to any given decimal place,by asking for

```
INT(X*10↑D+.5)/10↑D
```

where D is the number of decimal places desired, as in the following program.

The INT function is exercised in the following program (INPUT, PRINT, and GOTO statements are explained in Sections 3.3, 3.4, and 4.3.1 respectively).

```
10 REM - INT FUNCTION EXAMPLE
20 PRINT "NUMBER TO BE ROUNDED";
30 INPUT A
40 PRINT "NO. OF DECIMAL PLACES";
50 INPUT D
60 LET B=INT(A*10↑D+.5)/10↑D
70 PRINT "A ROUNDED ="B
80 GOTO 20
90 END

RUN
NUMBER TO BE ROUNDED? 55.65342
NO. OF DECIMAL PLACES? 2
A ROUNDED = 55.65
NUMBER TO BE ROUNDED? 78.375
NO. OF DECIMAL PLACES? -2
A ROUNDED = 100
NUMBER TO BE ROUNDED? 67.89
NO. OF DECIMAL PLACES? -1
A ROUNDED = 70
NUMBER TO BE ROUNDED?                    (The S key was typed here.
STOP                                     See section 6.4.)

READY
```

For negative numbers the largest integer contained in the number is a negative number with the same or a larger absolute value. For example:

```
INT(-23) = -23
```

but

```
INT(-14.39) = -15
```

2.8.3 Truncation Function, FIX(X)

The truncation function returns the integer part of X. For example:

```
FIX(10.2) = 10
```

and

```
FIX(-11.001) = -11
```

Notice that FIX is like INT for positive arguments. In fact, FIX could be defined as:

```
FIX(X) = SGN(X)*INT(ABS(X))
```

2.8.4 Random Number Function, RND(X)

The random number function produces a random number between 0 and 1. The numbers are reproducible in the same order for later checking of a program. The argument X in the RND(X) function call can be any number, as that value is ignored and serves no function.

The RND function is demonstrated in the following program (FOR and NEXT statements are explained in Sections 4.2.1 and 4.2.2 respectively).

```
    1Ø REM - RANDOM NUMBER EXAMPLE
    25 PRINT "RANDOM NUMBERS"
    3Ø FOR I=1 TO 3Ø
    4Ø PRINT RND(Ø),
    5Ø NEXT I
    6Ø END

RUN
RANDOM NUMBERS
 .2431684         .2988412         .7295ØØ8         .3125257         .3Ø95865
 .Ø4493979        .4834217         .4961Ø24         .5Ø1ØØ26         .Ø41Ø3271
 .2373254         .3Ø46887         .1923863         .9121199         .241212
 .9882844         .2587987         .Ø3323189        .87Ø1425         .9218898
 .8252Ø91         .2793Ø76         .374Ø252         .23Ø4741         .2666176
 .9ØØ4Ø82        2.951853E-3       .Ø39Ø6796        .7Ø8Ø242         .1465Ø3

READY
```

In order to obtain random digits from 0 to 9, change line 40 to read:

```
4Ø PRINT INT(1Ø*RND(Ø)),
```

and tell BASIC to run the program again. This time the results will look as follows:

```
RUN
RANDOM NUMBERS
 2                2                7                3                3
 Ø                4                4                5                Ø
 2                3                1                9                2
 9                2                Ø                8                9
 8                2                3                2                2
 9                Ø                Ø                7                1

READY
```

It is possible to generate random numbers over any range. For example, if the range (A,B) is desired, use:

```
        (B-A) * RND(Ø) + A
```

to produce a random number in the range A<n<B.

2.8.4.1 Randomize Statement

If you want the random number generator to calculate different random numbers every time the program is run, BASIC provides the RANDOMIZE statement. RANDOMIZE is normally placed at the beginning of a program which uses random numbers (the RND function). When executed, RANDOMIZE causes the RND function to choose a random starting value, so that the same program run twice will give different results. For example:

```
        1Ø RANDOMIZE
        2Ø PRINT RND(Ø)
        3Ø END
```

will print a different number each time it is run. For this reason, it is good practice to debug a program completely before inserting the RANDOMIZE statement.

The form of the statement is as follows:

```
            line-number RANDOMIZE
        or
            line-number RANDOM          (abbreviated form)
```

To demonstrate the effect of the RANDOMIZE statement on two runs of the same program, we insert the RANDOMIZE statement as line 15 below:

```
          15 RANDOM
          2Ø FOR I=1 TO 5
          25 PRINT "VALUE" I "IS" RND(Ø)
          3Ø NEXT I
          35 END
        RUN
        VALUE 1 IS .8Ø56684
        VALUE 2 IS .9863412
        VALUE 3 IS .542ØØØ8
        VALUE 4 IS .75ØØ257
        VALUE 5 IS .622Ø865

        READY
```

```
RUN
VALUE 1 IS .6914106
VALUE 2 IS .6435677
VALUE 3 IS .5136805
VALUE 4 IS .6650648
VALUE 5 IS .3672036

READY
```

Clearly, the output from each run is different.

2.8.5 User-defined Functions, DEF

In some programs, it may be necessary to execute the same mathematical formula in several different places. BASIC allows the programmer to define his own functions and call these functions in the same way he would call the square root or trig functions.

These user-defined functions are written in a defining or DEF statement. The function name consists of a three-letter name, the first two letters of which must be FN. The DEF statement is formed as follows:

line-number DEF FNA(X) = formula using X

where A is any letter. For example:

```
10 DEF FNA(S) = S↑2
```

will cause a later statement:

```
20 LET R = FNA(4)+1
```

to be evaluated as R=17.

The two following programs

Program #1:

```
10 DEF FNX(A) = A↑A
20 FOR I=1 TO 5
30 PRINT I, FNX(I)
40 NEXT I
50 END
```

Program #2:

```
10 DEF FNS(X) = X↑X
20 FOR I=1 TO 5
30 PRINT I, FNS(I)
40 NEXT I
50 END
```

both cause the same output:

```
RUN
 1              1
 2              4
 3              27
 4              256
 5              3125

READY
```

The argument in the DEF statement can be seen to have no significance; it is strictly a dummy variable. The function itself can be defined in the DEF statement in terms of numbers, dummy variables, real variables, other functions, or mathematical expressions. For examples:

```
10 DEF FNA(X) = X↑2+3*X+4

15 DEF FNB(X) = FNA(X)/2 + FNA(X)

20 DEF FNC(X) = SQR(X+4) + 1
```

The statement in which the user-defined function appears may have that function combined with numbers, variables, other functions, or mathematical expressions. For example:

```
40 LET R = FNA(X+Y+Z)
```

The user-defined function can be a function of more than one variable, as shown below:

```
25 DEF FNL(X,Y,Z) = SQR(X↑2 + Y↑2 + Z↑2)
```

A later statement in a program containing the above user-defined function might look like the following:

```
55 LET R = FNL(D,L,R)
```

where D, L, and R have some values in the program.

```
  1 REM MODULUS ARITHMETIC
 10 REM FIND X MOD M
 11 DEF FNM(X,M)=X-M*INT(X/M)
 12 REM
 20 REM FIND A+B MOD M
 21 DEF FNA(A,B,M)=FNM(A+B,M)
 22 REM
 30 REM FIND A*B MOD M
 31 DEF FNB(A,B,M)=FNM(A*B,M)
100 PRINT "ADDITION AND MULTIPLICATION TABLES MOD M"
110 PRINT "GIVE ME AN M";
120 INPUT M
140 PRINT "ADDITION TABLES MOD";M
150 GOSUB 800
200 FOR I=0 TO M-1
205 PRINT I;" ";
210 FOR J=0 TO M-1
220 PRINT FNA(I,J,M);
230 NEXT J
240 PRINT
250 NEXT I
260 PRINT
270 PRINT
280 PRINT "MULTIPLICATION TABLES MOD" M
290 GOSUB 800
300 FOR I=0 TO M-1
305 PRINT I;" ";
310 FOR J=0 TO M-1
320 PRINT FNB(I,J,M);
330 NEXT J
340 PRINT
350 NEXT I
360 STOP
800 PRINT
810 PRINT TAB(4);0;
820 FOR I=1 TO M-1
830 PRINT I;
840 NEXT I
850 PRINT
860 FOR I=1 TO 3*M+5
870 PRINT "-";
880 NEXT I
890 PRINT
900 RETURN
999 END
```

Figure 2-2 Modulus Arithmetic Program

(continued on next page)

```
RUN
ADDITION AND MULTIPLICATION TABLES MOD M
GIVE ME AN M? 7
ADLITION TABLES MOD 7

      Ø   1   2   3   4   5   6
----------------------------
Ø     Ø   1   2   3   4   5   6
1     1   2   3   4   5   6   Ø
2     2   3   4   5   6   Ø   1
3     3   4   5   6   Ø   1   2
4     4   5   6   Ø   1   2   3
5     5   6   Ø   1   2   3   4
6     6   Ø   1   2   3   4   5

MULTIPLICATION TABLES MOD 7

      Ø   1   2   3   4   5   6
----------------------------
Ø     Ø   Ø   Ø   Ø   Ø   Ø   Ø
1     Ø   1   2   3   4   5   6
2     Ø   2   4   6   1   3   5
3     Ø   3   6   2   5   1   4
4     Ø   4   1   5   2   6   3
5     Ø   5   3   1   6   4   2
6     Ø   6   5   4   3   2   1

READY
```

Figure 2-2. Modulus Arithmetic Program

$\searrow$ ≡ SH-L   Allows multiple statements in 1 line.

CHAPTER 3

INPUT/OUTPUT STATEMENTS

One of the most important groups of statements is the group of Input/Output (I/O) statements. These I/O statements allow us to bring data into our programs during execution when and from where we choose. Similarly, we can choose the output format which best suits our needs. In the case of the example program in Figure 2-1 (in Chapter 2), data was typed in at the console keyboard as BASIC requested it.

3.1 READ AND DATA STATEMENTS

A simple way to put data into a program is with READ and DATA statements. One statement is never used without the other. The READ statement is of the form:

line-number   READ   variables separated by commas

For example:

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

where A, B, and C are the variables we wish to assign values. In order to ensure that all variables are assigned values before computation begins, READ statements are usually placed at the beginning of a program, or at least before the point where the value is required for some computation.

Now that we have told BASIC to read the values for three variables, we must supply those values in a DATA statement of the form:

line-number DATA numeric values separated by commas

For example:

```
70 DATA 1,2,3
```

The DATA statement provides the values for the variables in the READ statement(s). The values must be separated by commas, in the same order as the variables are listed in the READ statement. Thus, at execution time, A=1, B=2, and C=3 according to lines 10 and 70 in the examples.

The DATA statement is usually placed at the end of a program before the END statement, so as to be easily accessible to the programmer should he wish to change his values.

A given READ statement may have more or fewer variables than there are values in any one DATA statement. READ causes BASIC to search all available DATA statements, in the order of their line numbers until all values are found for each variable. A second READ statement will begin reading values where the first stopped. If at some point in your program you attempt to read data which is not present, BASIC will stop and print an ERROR 47 message (see Appendix D) at the console, indicating the line which caused the error.

3.2 RESTORE STATEMENT

If it should become necessary to use the same data more than once in a program, the RESTORE statement will make it possible to recycle through the DATA statements beginning with the lowest numbered DATA statement. The RESTORE statement is of the form:

```
line-number   RESTORE
```

For example:

```
85 RESTORE
```

will cause the next READ statement following line 85 to begin reading data from the first DATA statement in the program, regardless of where the last data value was found.

You may use the same variable names the second time through the data or not, as you choose, since the values are being read as though for the first time. In order to skip unwanted values, dummy variables must be read. In the following example, BASIC prints:

```
 4              1              2              3
```

on the last line because it did not skip the value for the original N when it executed the loop beginning at line 210.

```
 10 REM - PROGRAM TO ILLUSTRATE USE OF RESTORE
 20 READ N
 25 PRINT "VALUES OF X ARE:"
 30 FOR I=1 TO N
 40 READ X
 50 PRINT X,
 60 NEXT I
 70 RESTORE
185 PRINT
190 PRINT "SECOND LIST OF X VALUES"
200 PRINT "FOLLOWING RESTORE STATEMENT:"
210 FOR I=1 TO N
220 READ X
230 PRINT X,
240 NEXT I
250 DATA 4,1,2
251 DATA 3,4
300 END

READY

RUN
VALUES OF X ARE:
 1              2              3              4
SECOND LIST OF X VALUES
FOLLOWING RESTORE STATEMENT:
 4              1              2              3
READY
```

3.3 INPUT STATEMENT

Another way to input data to a program is with an INPUT statement. This statement is used when writing a program to process data which is to be supplied while the program is running. The programmer types in the values as BASIC asks for them. Depending upon how many values are to be brought in by the input command, the programmer may wish to write himself a note reminding himself what data is to be typed in at what time. In the example program in Figure 3-1, the question is asked at execution time INTEREST IN PERCENT?, AMOUNT OF LOAN?, and NUMBER OF YEARS? The programmer knows which value is requested and proceeds to type and enter the appropriate number.

The INPUT statement is of the form:

line-number INPUT variables separated by commas

For example:

```
10 INPUT A,B,C
```

```
 10 REM - PROGRAM TO COMPUTE INTEREST PAYMENTS
 20 PRINT "INTEREST IN PERCENT";
 25 INPUT J
 26 LET J=J/100
 30 PRINT "AMOUNT OF LOAN";
 35 INPUT A
 40 PRINT "NUMBER OF YEARS";
 45 INPUT N
 50 PRINT "NUMBER OF PAYMENTS PER YEAR";
 55 INPUT M
 60 LET N=N*M
 65 LET I=J/M
 70 LET B=1+I
 75 LET R=A*I/(1-1/B↑N)
 78 PRINT
 80 PRINT "AMOUNT PER PAYMENT =";R
 85 PRINT "TOTAL INTEREST     =";R*N-A
 90 LET B=A
 95 PRINT " INTEREST       APP TO PRIN    BALANCE"
100 LET L=B*I
110 LET P=R-L
120 LET B=B-P
130 PRINT L,P,B
140 IF B>=R GOTO 100
150 PRINT B*I,R-B*I
160 PRINT "LAST PAYMENT =" B*I+B
200 END

READY

RUN
INTEREST IN PERCENT? 9
AMOUNT OF LOAN? 2500
NUMBER OF YEARS? 2
NUMBER OF PAYMENTS PER YEAR? 4

AMOUNT PER PAYMENT = 344.9617
TOTAL INTEREST     = 259.6932
 INTEREST       APP TO PRIN    BALANCE
 56.25          288.7117       2211.288
 49.75399       295.2077       1916.081
 43.11182       301.8498       1614.231
 36.3202        308.6415       1305.589
 29.37576       315.5859       990.0036
 22.27508       322.6866       667.317
 15.01463       329.947        337.37
 7.590824       337.3708
LAST PAYMENT = 344.9608

READY
```

Figure 3-1   Interest Program

will cause BASIC to pause during execution, print a question mark, and wait for the user to type in three numerical values separated by commas and followed by typing the RETURN key at the end of the list.

As you will notice in Figure 3-1, the question mark is grammatically useful if you care to formulate a verbal question which the input value will answer. This will be further explained in the next section (the PRINT Statement).

The output for the program begins after the word RUN and includes a verbal description of the numbers. This verbal description on the output is optional with the programmer, although it has a definite advantage in ease of use and understanding.

When the correct number of variables have been typed, type the RETURN key to enter them to BASIC.

If too few values are typed (entered), another question mark will be printed.

If too many values are typed, the message:

```
ERROR 45 IN XXXX
```

will be given. (XXXX represents the line number of the INPUT statement.)

3.4 PRINT STATEMENT

The PRINT statement is the output statement for BASIC. Depending upon what follows the PRINT command, we can create different output formats and even plot points on a graph.

In order to skip a line on the output sheet, type only a line number and the command PRINT:

```
10 PRINT
```

When BASIC comes to line 10 during the run, the paper in the Teletype will be advanced by one line. In the example program in Figure 3-1, line 78 causes a blank line on the output sheet between the section where the user enters data to BASIC and the section where BASIC supplies the results of the program.

In order to have BASIC print the results of a computation or the value of any variable at any point in the program, the user types the line number, the command PRINT, and the variable names, separated by commas:

```
10 PRINT A,  C+B,  SQR(A)
```

This will cause the values of A,C+B, and the square root of A to be printed in the first three of the five fixed format columns (of 14 spaces each) which BASIC uses for most output. For example, the statement:

```
10 PRINT A,B,C,D,E
```

will cause the values of the variables to be printed like this:

```
12.3          12.3          12.3          12.3          12.3
```

where A, B, C, D, E equal 12.3. When more than five variables are listed in the PRINT statement and separated by commas, the sixth value begins a new line of output.

The third possibility for the PRINT statement is to print a message or some text. The user may have a message printed by placing the message in quotation marks. For example:

```
10 PRINT "THIS IS A TEST"
```

When line 10 above is encountered during execution, the following will be printed:

```
THIS IS A TEST
```

Go back to the example program in Figure 3-1 and notice the function of lines 80, 85, and 95.

Figure 3-1 shows that the PRINT statement can combine the second and third options above. One PRINT command tells the computer to print:

```
AMOUNT PER PAYMENT = 344.9617
```

The command which did this was line 80:

```
80 PRINT "AMOUNT PER PAYMENT =";R
```

PRINT CHR$($x_8$) → Char ≡ x_8

It is not necessary to use the standard five-column format for output. A semi-colon (;) will cause the following text or data to be printed following the last character of text or data printed. A comma (,) will cause a jump to the next of the five output format columns. BASIC allows the user to omit format control characters (,) or (;) between text and data, and assumes a semi-colon. For example:

```
PRINT "AMOUNT PER PAYMENT ="R
```

will result in the same output as line 80 above.

If you wish to skip over one of the five fixed format columns, simply use two successive commas. For example:

```
   5 LET A = 100
  10 LET B = 100
  20 PRINT A,,B
  25 END

RUN
 100                              100

READY
```

In addition to the capabilities already mentioned, the PRINT statement can also cause a constant to be printed at the console. For example:

```
  10 PRINT 1.234, SQR(100/4)
  15 END
```

will cause the following to be printed at execution time:

```
 1.234           5
```

Any constant present in a PRINT statement will be printed exactly as shown. Any algebraic expression in a PRINT statement will be evaluated with the current value of the variables and the result printed.

In Figure 3-1, line 160 reads:

```
 160 PRINT "LAST PAYMENT =" B*I+B
```

and caused the following to be printed upon execution:

```
LAST PAYMENT = 344.9608
```

This demonstrates the omission of the format control character as well as the ability of the PRINT statement to print text and do calculations.

The following example program illustrates the use of the control characters in PRINT statements:

```
10 READ A,B,C
20 PRINT A,B,C,A↑2,B↑2,C↑2
30 PRINT
40 PRINT A;B;C;A↑2;B↑2;C↑2
50 DATA 4,5,6
60 END

READY

RUN
 4              5              6              16             25
 36

 4  5  6  16  25  36

READY
```

If a number should happen to be too long to be printed on the end of a single line, BASIC automatically moves the entire number to the beginning of the next line.

3.5 TAB FUNCTION

When using the PRINT statement thus far we have had to print a blank character wherever we wanted blank space; there was no real control over printing. The TAB function is a more sophisticated technique allowing the user to position the printing of characters anywhere on the Teletype paper line. This line is 72 characters long, and the print positions can be thought of as being numbered from 0 to 71, going from left to right. The TAB function argument can be positive or negative: TAB(-1) causes a tab over to position 71, TAB(3) causes a tab to position 3. (The TAB function can be thought of as operating module 72.)

After performing TAB(n), the next character to be printed will be placed in position n. If n is a position to the left of the current position, a carriage return without a line feed is performed to correctly position the printing head. For example:

```
10 PRINT "X =";TAB(2);"/";3.14159
15 END
```

will print the slash on top of the equal sign, as shown below:

```
RUN
X ≠ 3.14159
```

The following is an example of the sort of graph that can be drawn with BASIC using the TAB function.

```
  30 FOR X=0 TO 15 STEP .5
  40 PRINT TAB(30+15*SIN(X)*EXP(-.1*X));"*"
  50 NEXT X
  60 END

RUN
                              *
                                    *
                                         *
                                          *
                                         *
                                    *
                               *
                          *
                      *
                    *
                     *
                       *
                           *
                               *
                                  *
                                    *
                                    *
                                   *
                                *
                             *
                          *
                         *
                         *
                         *
                           *
                             *
                               *
                                 *
                                 *
                                 *
                                *

READY
```

CHAPTER 4

SUBSCRIPTS AND LOOPS

4.1 SUBSCRIPTED VARIABLES

In addition to simple variable names, there is a second class of variables which BASIC accepts, called subscripted variables. Subscripted variables provide the programmer with additional computing capabilities for dealing with lists, tables, matrices, or any set of related variables. In BASIC variables are allowed one or two subscripts. A single letter or a letter followed by a digit forms the name of the variable; followed by one or two formulas in parentheses, separated by commas, indicating the place of that variable in the list. For example, a list might be described as A(I) where I goes from 1 to 5 as shown below:

```
A(1),A(2),A(3),A(4),A(5)
```

This allows the programmer to reference each of the five elements in the list A. A two dimensional matrix A(I,J) can be defined in a similar manner, but the subscripted variable A can only be used once. A(I) and A(I,J) cannot be used in the same program.

It is possible, however, to use the same variable name as both a subscripted and as an unsubscripted variable. Both A and A(I) are valid variable names and can be used in the same program.

Any formula (number, variable, function, or combination) can be a subscript. For example:

```
10 LET A=1
20 LET B=3
30 LET D=C(A+1,B)
```

is the same as:

```
10 LET D=C(2,3)
```

If the formula is non-integer after evaluation, the value is truncated to an integer. For example:

```
10 LET A=3
20 LET B=C(1+A/2)
```

is the same as:

```
10 LET B=C(2.5)
```

or

```
10 LET B=C(2)
```

In other words subscripts are like :

```
A(X,Y) = A(FIX(X),FIX(Y))
```

The following program illustrates the use of subscripted variables.

```
10 REM - PROGRAM DEMONSTRATING READING OF
11 REM - SUBSCRIPTED VARIABLES
15 DIM A(5), B(2,3)
18 PRINT "A(I) WHERE A=1 TO 5:"
20 FOR I=1 TO 5
25 READ A(I)
30 PRINT A(I);
35 NEXT I
38 PRINT
39 PRINT
40 PRINT "B(I,J) WHERE I=1 TO 2"
41 PRINT "           AND J=1 TO 3:"
42 FOR I=1 TO 2
43 PRINT
44 FOR J=1 TO 3
48 READ B(I,J)
50 PRINT B(I,J);
55 NEXT J
56 NEXT I
60 DATA 1,2,3,4,5,6,7,8
61 DATA 8,7,6,5,4,3,2,1
65 END
```

```
RUN
A(I) WHERE A=1 TO 5:
 1  2  3  4  5

B(I,J) WHERE I=1 TO 2
          AND J=1 TO 3:

 6  7  8
 8  7  6
READY
```

4.1.1 DIM Statement

As in the preceding examples, we see that with the use of subscripts we used a dimension (DIM) statement:

Since EduSystem 20 BASIC defines all variables as they occur, DIM statements are ignored. For example:

```
  10 REM - MATRIX CHECK PROGRAM
  20 FOR I=0 TO 6
  22 LET A(I,0) = I
  25 FOR J=0 TO 10
  28 LET A(0,J) = J
  30 PRINT A(I,J);
  35 NEXT J
  40 PRINT
  45 NEXT I
  50 END

RUN
 0  1  2  3  4  5  6  7  8  9  10
 1  0  0  0  0  0  0  0  0  0  0
 2  0  0  0  0  0  0  0  0  0  0
 3  0  0  0  0  0  0  0  0  0  0
 4  0  0  0  0  0  0  0  0  0  0
 5  0  0  0  0  0  0  0  0  0  0
 6  0  0  0  0  0  0  0  0  0  0

READY
```

Notice that a variable has a value of zero until it is assigned a value.

The limits on subscript size are as follows:

1. Single subscript: 0 to 2047
2. Double subscripts: 0 to 63 for each subscript

4.2 LOOPS

So far in these chapters we have seen FOR and NEXT statements used several times in examples. These two statements define the beginning and end of a loop. A loop is a set of instructions which modifies itself and repeats until some terminal condition is reached.

4.2.1 FOR Statement

The FOR statement is of the form:

```
line-number  FOR  variable=formula TO formula STEP formula
```

For example:

```
10 FOR K=2 TO 20 STEP 2
```

which will iterate (cycle) through the designated loop using K as 2, 4, 6, 8,...,20. in calculations involving K. When the value 20 is reached, the loop is left behind and the program goes to the line following the NEXT statement (described in Section 4.2.2).

The variable mentioned in the definition must be unsubscripted, although a common use of such loops is to deal with subscripted variables using the FOR variable as the subscript of a previously defined variable. The formulas mentioned in the definition can be real or integer numbers, variables, or expressions.

4.2.2 NEXT Statement

The NEXT statement signals the end of the loop and at that point BASIC adds the STEP value to the variable and checks to see if the variable is still less than or equal to the terminal value. When, after incrementing, the variable exceeds the terminal value, control falls through the loop to the following statement. If the STEP size is negative, control falls through the loop when the variable is less than the terminal value.

When control falls through the loop, the variable value is one step beyond what it was when the loop was last executed. For some programs, this information may be useful.

If the STEP value is omitted, +1 is assumed. Since +1 is the usual STEP value, that portion of the statement is frequently omitted.

In the following example, we see a demonstration of the second and third paragraphs of this section. The loop is executed 10 times, the value of I is 11 when control leaves the loop, and +1 is the assumed STEP value.

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

RUN
 11

READY
```

If line 10 had been:

```
10 FOR I=10 TO 1 STEP -1
```

the value printed by BASIC would be 0.

The numbers used in the FOR statement can be formulas as indicated earlier. A formula, in this case, can be a variable, a mathematical expression, or a numerical value.

The value of each formula is evaluated upon first encountering the loop. While the values of the variables, if any, used in evaluating these formulas can be changed within the loop, the values assigned in the FOR statement remain as they were initially defined.

In the last example program the value of I in line 10 can be successfully changed in the program. The loop:

```
10 FOR I=1 TO 10
15 LET I=10
20 NEXT I
```

will only be executed once, since the value 10 has been reached by the variable I and the termination condition is satisfied.

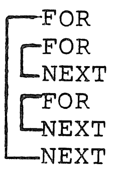
4.2.3 Nesting Loops

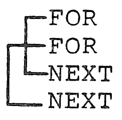
It is often useful to have one or more loops within a loop. This technique is called nesting. Nesting is allowed as long as the field of one loop (the numbered lines from the FOR statement to the corresponding NEXT statement, inclusive) does not cross the field of another loop. A diagram is the best way to illustrate acceptable nesting procedures:

ACCEPTABLE NESTING TECHNIQUES

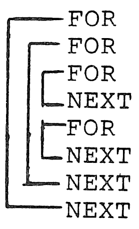
UNACCEPTABLE NESTING TECHNIQUES

Two Level Nesting





Three Level Nesting



If the value of the counter variable is originally set equal to the terminal value, the loop will execute once, regardless of the STEP value.

It is also possible to exit from a FOR-NEXT loop without the counter variable reaching the termination value. A conditional transfer may be used to leave a loop. Control may only transfer <u>into</u> a loop which had been left earlier without being completed, ensuring that the termination and STEP values are assigned.

4.3 TRANSFER OF CONTROL

Certain statements can cause the execution of a program to jump to a different line either unconditionally or depending upon some condition within the program. Looping is one method of jumping to a designated point until a condition is met. The following commands give the programmer additional capabilities in this area.

4.3.1 Unconditional Transfer, GOTO

The GOTO statement is an unconditional command telling BASIC to jump either ahead or back in the program. For example:

```
100 GOTO 50
```

or

```
24 GOTO 78
```

The GOTO statement is of the form:

```
line-number   GOTO   line-number
```

When the logic of the program reaches the GOTO statement, the statement(s) immediately following it will not be executed, but the statements beginning with the line number indicated are performed.

The program below never ends; it does a READ, prints something, and attempts to do this over and over until it runs out of data, which is sometimes an acceptable (though not advisable) way to end a program.

```
  10 REM - PROGRAM ENDING WITH ERROR
  11 REM - MESSAGE WHEN OUT OF DATA
  20 READ X
  25 PRINT "X="X,"X↑2="X↑2
  30 GOTO 20
  35 DATA 1,5,10,15,20,25
  40 END

READY

RUN
X= 1           X↑2= 1
X= 5           X↑2= 25
X= 10          X↑2= 100
X= 15          X↑2= 225
X= 20          X↑2= 400
X= 25          X↑2= 625

ERROR  47 IN   20

READY
```

4.3.2 Conditional Transfer, IF-THEN, IF-GOTO, ON-GOTO

If a program requires that two values be compared at some point, logic may direct us to different procedures depending on the comparison. In computing, we logically test values to see whether they are equal, greater, or less than another value, or a possible combination of the three.

In order to compare values, we use a group of mathematical symbols not discussed earlier. These symbols are as follows:

BASIC Symbol	Math Symbol	BASIC Example	Meaning
=	$=$	A = B	A is equal to B
<	$<$	A < B	A is less than B
<=	$\leq$	A <=B	A is less than or equal to B
>	$>$	A > B	A is greater than B
>=	$\geq$	A >=B	A is greater than or equal to B
<>	$\neq$	A <>B	A is not equal to B

4.3.2.1 IF-THEN and IF-GOTO

The IF-THEN and IF-GOTO statements both allow the programmer to test the relationship between two formulas (variables, numbers, or expressions). Providing the relationship we have described in the IF statement is true at that point, control will transfer to the line number indicated. The statements are of the form:

line-number IF formula relation formula

$\left\{ \begin{matrix} \text{THEN} \\ \text{GOTO} \end{matrix} \right\}$ line-number

The use of the word THEN or GOTO is the programmer's choice.

For example:

```
10 IF A=5 GOTO 70
```

causes transfer to line 70 if A is equal to 5. If A is not equal to 5, control passes to the next line of the program following line 10.

4.3.2.2 ON-GOTO

The ON-GOTO statement permits the program to transfer control to one of a set of lines depending on the value of a formula. The statement is of the form

line-number ON formula GOTO line-number, line-number ...

The formula is evaluated and then truncated to an integer. This integer is used as index to tell which of the line numbers to transfer control to. If the integer is one, then the first line number is used. If it is two, then the second is used. Similarly for high values. Obviously, the formula after truncation cannot be zero or negative or greater than the number of line numbers in the list. For example:

```
10 ON A+2 GOTO 100,200,300,400
```

If A is 2 then control is passed to line 400. The range A can have in this example is -1 to 2.

CHAPTER 5

SUBROUTINES

When particular mathematical expressions are evaluated several times throughout a program, the DEF statement enables the user to write that expression only once. The technique of looping allows the program to do a sequence of instructions a specified number of times. If the program should require that a sequence of instructions be executed several times in the course of the program, this too is possible.

A subroutine is a section of code performing some operation that is required at more than one point in the program. Sometimes a complicated I/O operation for a volume of data, a mathematical evaluation which is too complex for a user-defined function, or any number of other processes may be best performed in a subroutine.

5.1 GOSUB STATEMENT

Subroutines are placed physically at the end of a program, usually before DATA statements, if any, and always before the END statement. The program begins execution and continues until it encounters a GOSUB statement of the form:

```
line-number  GOSUB  line-number
```

where the number after GOSUB is the first line number of the subroutine; control then transfers to that line in the subroutine. For example:

```
50 GOSUB 200
```

5.2 RETURN STATEMENT

Having reached line 50, as shown on the previous page, control transfers to line 200; the subroutine is processed until BASIC encounters a RETURN statement of the form:

```
line-number  RETURN
```

which causes control to return to the line following the GOSUB statement. Before transferring to the subroutine, BASIC internally records the next line number to be processed after the GOSUB statement; the RETURN statement is a signal to transfer control to this line. In this way, no matter how many subroutines or how many times they are called, BASIC always knows where to go next. The following program demonstrates a simple subroutine:

```
  1 REM - THIS PROGRAM ILLUSTRATES GOSUB AND RETURN
 10 DEF FNA(X)= ABS(INT(X))
 20 INPUT A,B,C
 30 GOSUB 100
 40 LET A=FNA(A)
 50 LET B=FNA(B)
 60 LET C=FNA(C)
 70 PRINT
 80 GOSUB 100
 90 STOP
100 REM - THIS SUBROUTINE PRINTS OUT THE SOLUTIONS
110 REM - OF THE EQUATION: AX↑2 + BX + C = 0
120 PRINT "THE EQUATION IS " A " *X↑2 + " B " * X + " C
130 LET D= B*B - 4*A*C
140 IF D<>0 THEN 170
150 PRINT "ONLY ONE SOLUTION... X =" -B/(2*A)
160 RETURN
170 IF D<0 THEN 200
180 PRINT "TWO SOLUTIONS... X =";
185 PRINT (-B+SQR(D))/(2*A) "AND X =" (-B-SQR(D))/(2*A)
190 RETURN
200 PRINT "IMAGINARY SOLUTIONS... X = (";
205 PRINT -B/(2*A) "," SQR(-D)/(2*A) ") AND (";
207 PRINT -B/(2*A) "," -SQR(-D)/(2*A) ")"
210 RETURN
900 END

RUN
? 1,.5,-.5
THE EQUATION IS  1  *X↑2 +  .5  * X + -.5
TWO SOLUTIONS... X = .5 AND X =-1

THE EQUATION IS  1  *X↑2 +  0  * X +  1
IMAGINARY SOLUTIONS... X = ( 0 , 1 ) AND ( 0 ,-1 )

READY
```

Lines 100 through 210 constitute the subroutine. The subroutine is executed from line 30 and again from line 80. When control returns to line 90 the program encounters the STOP statement and terminates execution. Note that even though the program logically ends with a STOP, the END command must still be present.

For another detailed example of a subroutine, see Figure 3-1.

5.3 STOP AND END STATEMENTS

The STOP statement is used synonymously with the END statement to terminate execution, but the END statement must be the last statement of the entire program. STOP may occur several times throughout the program. No BASIC program will run without an END statement of the form:

```
        line-number  END
```

The format of the STOP statement is simply:

```
        line-number  STOP
```

STOP is equivalent to a GOTO nn, where nn is the line number of the END statement.

5.4 NESTING SUBROUTINES

More than one subroutine can be used in a single program, in which case they can be placed one after another at the end of the program (in line number sequence). A useful practice is to assign distinctive line numbers to subroutines; for example, if you have numbered the main program with line numbers up to 199, you could use 200 and 300 as the first numbers of two subroutines.

Subroutines can also be nested, in terms of one subroutine calling another subroutine. If the execution of a subroutine encounters a RETURN statement, it will return control to the line following the GOSUB which called that subroutine; therefore, a subroutine can call another subroutine, even itself. Subroutines can be entered at any point and have more than one RETURN statement, where certain conditions will cause control to reach any one RETURN statement. It is possible to transfer to the beginning or any part of a subroutine; multiple entry points and RETURNS make a subroutine more versatile.

CHAPTER 6

ERRORS AND HOW TO MAKE CORRECTIONS

6.1 SINGLE LETTER CORRECTIONS

Nobody being perfect, we all make typing errors if not logical errors. The first is by far the easier to correct. If you notice an error immediately as you type it, for example:

```
        10 LEB
```

instead of LET, as you meant to begin the line, type the RUBOUT key or SHIFT/O (back-arrow) once for every character you wish to remove, including spaces. This will result in the printing of a back-arrow to show that the rubout has been accomplished. Make the correction and continue typing as shown below.

```
        10 LEB←T A=10*B
```

if that was the intended line. BASIC does not even see the mistake; it is erased, except on the console as you typed it. The typed line enters the computer only when you type the RETURN key. Before that time, you can correct errors with the RUBOUT key or SHIFT/O. If you desire a neat, corrected listing at the end of your work, that is possible too. More on that later. For now, consider that:

```
        20 DEN F←←←F FNA(X,Y) = X↑2+3*Y
```

is the same as:

```
        20 DEF FNA(X,Y) = X↑2+3*Y
```

to BASIC. Notice that you erase spaces as well as print characters.

6.2 ERASING A LINE

If at any time you have typed a line and not yet typed the RETURN key, the line can be erased by typing the ALTMODE (ESCape on some machines) key. BASIC will echo:

```
        $ DELETED
```

at the end of the line to indicate that the line has been removed. You can continue typing on that line as though it were the start of a new line or type the RETURN key to start a fresh line.

Once you have typed the RETURN key and have entered a line into BASIC, the line can still be corrected simply by typing the line number and proceeding to retype the line correctly. The old line is automatically deleted as you type the line number again, even if it was longer than the new line.

You can delete an entire line by typing the line number and then the RETURN key. This removes the entire line and line number from your program.

Following an attempt to run a program, you may receive an error message. Most errors can be corrected by typing the line number, typing the line over again with the correction, and then typing the RETURN key. The program is then ready to be run again. You can make as many changes or corrections between runs as you wish. (For a more advanced technique in program editing, see Chapter 8.)

6.3 ERASING A PROGRAM IN CORE

Assuming you have written a program on-line in BASIC, have completed it, and now wish to run another program in BASIC but do not wish to save the old program, when BASIC prints READY, answer:

```
    SCRATCH
```

The SCRATCH command will erase the old program and leave a fresh, blank area in which you can work. Only the abbreviation SCR is necessary. BASIC will again reply READY, and you proceed from this point. You can, alternatively, reply to READY with NEW or BYE, which both clear core just as does SCRATCH.

If, after BASIC prints READY, you merely begin typing a new program without clearing core, BASIC will retain the previous program and, in effect, you will write over that program as though you were changing each single line. However, if you do not remove or type over <u>all</u> of the previous line numbers, you will discover the unchanged lines appearing in the new program as well. To avoid this, tell BASIC to SCRATCH the old program.

6.4 STOPPING A RUN

If your program begins to print what you know will be a long list of unwanted output for one reason or another, you can stop a running program by pressing the S key; STOP will be printed, and the program will stop execution, returning you to edit mode (BASIC prints READY).

CHAPTER 7

RUNNING A BASIC PROGRAM

7.1 RUN COMMAND

When your program is ready to be run (be sure there is an END statement), type RUN, then the RETURN key, and the program will attempt to execute. If there is some error in the way you wrote your BASIC statements, an error message will be printed, following which you may correct the errors one line at a time. Then type the RUN command again. If the program executes correctly, you will obtain whatever printed output you requested. When the END statement is reached, BASIC stops execution and again prints READY.

7.2 EDITING PHASE

To simplify matters, we can think of BASIC as operating in two phases: a run phase and an editing phase. The run phase is the interval between the time you type RUN and the time BASIC prints READY; this is the time during which BASIC is compiling and executing your program. Once BASIC has printed READY, it is able to accept commands directly from your keyboard; during this editing phase you can prepare your program and can direct BASIC to perform a variety of services such as the SCRATCH command. The commands can all be abbreviated to three letters, some have arguments, others do not, as explained below.

7.2.1 LIST Command

Once your program works, you may discover you have several feet of Teletype paper filled with corrections. To obtain a clean listing of your program, type LIST followed by the RETURN key. The whole program will be printed. You can then tell BASIC to RUN, and your output will follow.

For debugging purposes, it is sometimes useful to list part of your program. LIST or LIS, followed by one line number or two line numbers separated by a comma, will result in BASIC printing either that single line or the lines between and including the two numbers given.

7.2.2 DELETE Command

DELETE followed by two line numbers separated by a comma will cause all lines between and including the two given to be deleted from the program. If only one line number is given, that line will be deleted. For example:

```
        DELETE 10,20
```

causes all lines between 10 and 20 inclusive to be deleted.

7.2.3 ALTMODE Key

Typing the ALTMODE key (which prints a $) will cause any of the preceding commands (DELETE, LIST, SAVE, etc.) to be erased. ALTMODE must be typed before the RETURN key which enters the command into BASIC. If you do change your mind about a command, you can alter it as shown below:

```
        DELETE 10$ DELETED
```

BASIC replies $ DELETED to show that the command has been erased; you may then retype the line.

7.3 PUNCHING A PAPER TAPE

It may be useful in many cases to have a copy of a program you have written in BASIC stored on paper tape. You can create such a copy quite easily. Once you have completed your program to the point that you wish to copy it, punch a listing of it through BASIC. The steps involved are:

1. Type TAPE followed by the RETURN key. Any characters you type now will not echo on the console or on your tape.
2. Punch the ON button on the tape punch.
3. Type LIST followed by the RETURN key. This causes the program to be listed on paper tape and on the console.
4. Punch the OFF button on the tape punch.

Using LIST when in TAPE mode will result in the following:

1. The word LIST will not echo.
2. Leader tape is punched before and after the program.

You will notice that when you tear off the tape from the punch there will be an arrowhead on the tape; this shows the direction in which the tape is later to be inserted into the machine.

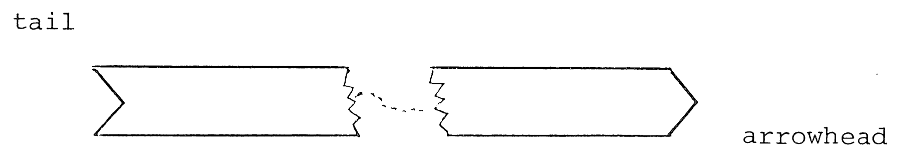


Figure 7-1   Paper Tape Diagram

Once you have finished punching your program, you will wish to return to regular operating mode. During TAPE mode, no characters you type will be echoed. Typing KEY followed by the RETURN key will bring you back to normal operating mode. You may then continue working on that program.

A paper tape can be duplicated by positioning the tape in the reader, depressing the ON button, turning the LINE-OFF-LOCAL knob to LOCAL, and turning the reader control switch to START.

7.4 READING AND LISTING A PAPER TAPE

To read in a paper tape from the low-speed reader on the Teletype, proceed as follows:

1. Position paper tape in the reader head:
   a. Raise retainer cover,
   b. Set reader control to FREE,
   c. Position paper tape with feed holes over the sprocket wheel, and the arrow (cut) pointing outward from the console.
   d. Close retainer cover to hold tape in position.
2. Type TAPE, then the RETURN key.
3. Set reader control switch to START until listing has been completed. Reader will not stop at blank tape. You must turn the reader control switch to FREE.
4. In order to get back into regular operating mode where the characters you type will be echoed at the console, type KEY and then the RETURN key.
5. BASIC will print READY; you can then ask BASIC to LIST and RUN your program.

CHAPTER 8

ADVANCED BASIC

This section deals with additional features of BASIC which, once you have learned the BASIC language, will make programming somewhat easier.

8.1 EDIT COMMAND

Frequently it is only necessary to correct several characters in a line. Rather than retype the entire line, which may be a complex formula or output format, there is a command which allows you to access a single line and search for the character you wish to change. The form of the EDIT command is as follows:

```
EDI  line-number
```

Notice that the EDIT command may be abbreviated to three letters. It is then followed by the line number of the statement to be changed. Enter the command by typing the RETURN key. At this point, BASIC waits for you to type a search character which BASIC will not print. The character you type will be some character which already exists on the line (one of the legal BASIC characters, ASCII 240 through 336 (excluding code 300,@) inclusive on the ASCII table in Appendix F). After the search character is typed, BASIC prints out the contents of that line until the search character is printed. At this point, printing stops and the user has the following options:

1. Type in new characters which are inserted following the ones already printed.
2. Type a Form Feed (CTRL/L); this will cause the search to proceed to the next occurrence, if any, of the search character.
3. Type a BELL (CTRL/G); this allows the user to change the search character. The user can specify a new search character.
4. Use the RUBOUT (or SHIFT/O) key to delete one character to the left each time RUBOUT is depressed. RUBOUT echoes as ←.
5. Type the RETURN key to terminate editing of the line at that point, removing any text to the right.
6. Type the ALTMODE key to delete all the characters to the left except the line number.
7. Type the LINE FEED key to terminate editing of the line, saving the remaining characters.

On completion of the EDIT operation, BASIC prints READY. Note that line numbers cannot be changed using EDIT, i.e., you cannot search for a line number digit.

The following example demonstrates the EDIT command where the incorrect line reads as follows:

```
        6Ø PRINT "PI=3.14146 ABOU*!"
```

To edit the line would result in the following output on the Teletype:

```
        EDIT 6Ø
        PRINT "PI=3.14146←←59 ABOU*←T!"

        LIST 6Ø

           6Ø PRINT "PI=3.14159 ABOUT!"

        READY
```

The operations involved in editing the line were as follows: First the number 6 was indicated as the search character. When the 6 was printed, the RUBOUT key was typed twice to remove the two incorrect digits and 59 was inserted in their places. CTRL/BELL was typed, resulting in BASIC accepting another search character. BASIC then prints to the search character * which is removed by a RUBOUT and replaced with T. The LINE FEED key was typed to terminate the edit and save the remaining characters.

8.2 COMMENTS AND REMARKS

You may append a comment to any line by starting the comment with a single apostrophe ('). All characters typed after the apostrophe are ignored when the program is run. For example:

```
        1Ø LET X=4 'SET X TO ITS INITIAL VALUE
        2Ø GOTO 1Ø 'LOOP BACK TO START
```

Within the print literals, the apostrophe is not considered as the start of a comment. For example:

```
        12 PRINT "X'S VALUE IS";X
```

which will print:

```
        X'S VALUE IS 4
```

Now consider the following.

```
        1Ø PRINT X,Y, "FINAL" 'LAST PRINTING
```

which will print:

```
        2              3              FINAL
```

because the single apostrophe introduced the comment.

8.3 INPUT STATEMENT

When responding to an INPUT statement, you may add a comment which will print on the Teletype but have no effect on the running program. For example:

```
    10 INPUT X
    20 PRINT X
    30 END

RUN
? 2 'LET X BE 2
 2

READY
```

A trailing comma may be appended to the response of an INPUT statement if it is not after the last value to be entered. For example:

```
    10 INPUT X,Y,Z
    20 PRINT X,Y,Z
    30 END

RUN
? 1,2,
? 3
 1              2              3

READY
```

However, the following is illegal.

```
RUN
? 1,2,3,

ERROR 45 IN   10

READY
```

8.4 ABBREVIATING COMMANDS

All commands and statement keywords can be abbreviated to the first three letters, as shown in Appendix A.

CHAPTER 9

LOADING, STARTING, AND INITIAL DIALOGUE

EduSystem 20 BASIC enables from one to five persons to simultaneously share the resources of EduSystem 20 computer hardware. The minimal System contains the following hardware:

One PDP-8, -8/L, or -8/I with 8K words of core memory

One to five ASR- or KSR-33 Teletypes and appropriate PT08 Units or DC02 Unit.

The System requires a minimum of 8K words of core; additional core, up to 28K, enhances the operation of the System. The addition of a High-Speed Paper Tape Reader/Punch is optional.

The System software is supplied on two paper tapes:

System Tape -- This tape contains the Binary Loader, the BASIC program, and BASIC's initial dialogue. It is loaded into core using the Read-In Mode (RIM) Loader (see Appendix E).

Configurator Tape -- This short tape contains the Binary Loader and BASIC's initial dialogue. It is used to respecify the system configuration (number of users, size of core, etc.) without having to reload the entire lengthy System tape. It is loaded into core using the RIM Loader.

9.1 LOADING AND STARTING BASIC

The procedure for loading either of the tapes follows.

1. Ensure that the RIM Loader is in Field 0 (see Appendix E).
2. Turn all Teletypes to LINE.
3. Set all Teletype tape readers to FREE.
4. Place the System or Configurator tape in the appropriate reader with leader code (ASCII 200) over the reader head.
5. Set the Data Field and Instruction Field to 0.
6. Set the Switch Register to 7756. (RIM starting address)
7. Press LOAD ADDress and then START.
8. If using low-speed reader, set Teletype tape reader to START.

The tape should start passing over the reader head. If not, check that the RIM Loader is in core correctly and that you are using the appropriate tape reader as determined by the RIM Loader program (low- or high-speed version); start over at step 1 above.

If the message EDUSYSTEM 20 BASIC does not print on the teleprinter when the tape stops passing through the reader, the tape was not read into core correctly. Therefore, check the RIM Loader and start over at step 1 above.

When EDUSYSTEM 20 BASIC is printed, continue at step 9.

9. Remove the tape from the tape reader.
10. Answer BASIC's initial dialogue as explained below.

9.2 BASIC'S INITIAL DIALOGUE

When started, BASIC prints an identification message and then commences to ask certain questions to which your response specifies the number of users and the amount of core to be used during operation. If you have made a mistake anywhere during the dialogue, simply type an S in place of one of the responses; BASIC will restart the dialogue.

The first question is printed two lines below the identification message:

```
EDUSYSTEM 2Ø  BASIC
NUMBER OF USERS(1 TO 5)?
```

to which your response should be a single digit from 1 to 5, depending on the number of Teletype terminals to be used. BASIC automatically prints the next question.

For example:

```
EDUSYSTEM 2Ø  BASIC

NUMBER OF USERS(1 TO 5)?4
HIGHEST CORE FIELD(1 TO 7)?
```

to which your response should be a single digit from 1 to 7, depending on the size of core in which BASIC is to operate. For a minimal 8K core, your response would be 1. Each digit is interpreted as follows:

1	8K core memory
2	12K core memory
3	16K core memory
4	20K core memory
5	24K core memory
6	28K core memory
7	32K core memory

The initial dialogue is diagrammed in Figure 9-1.

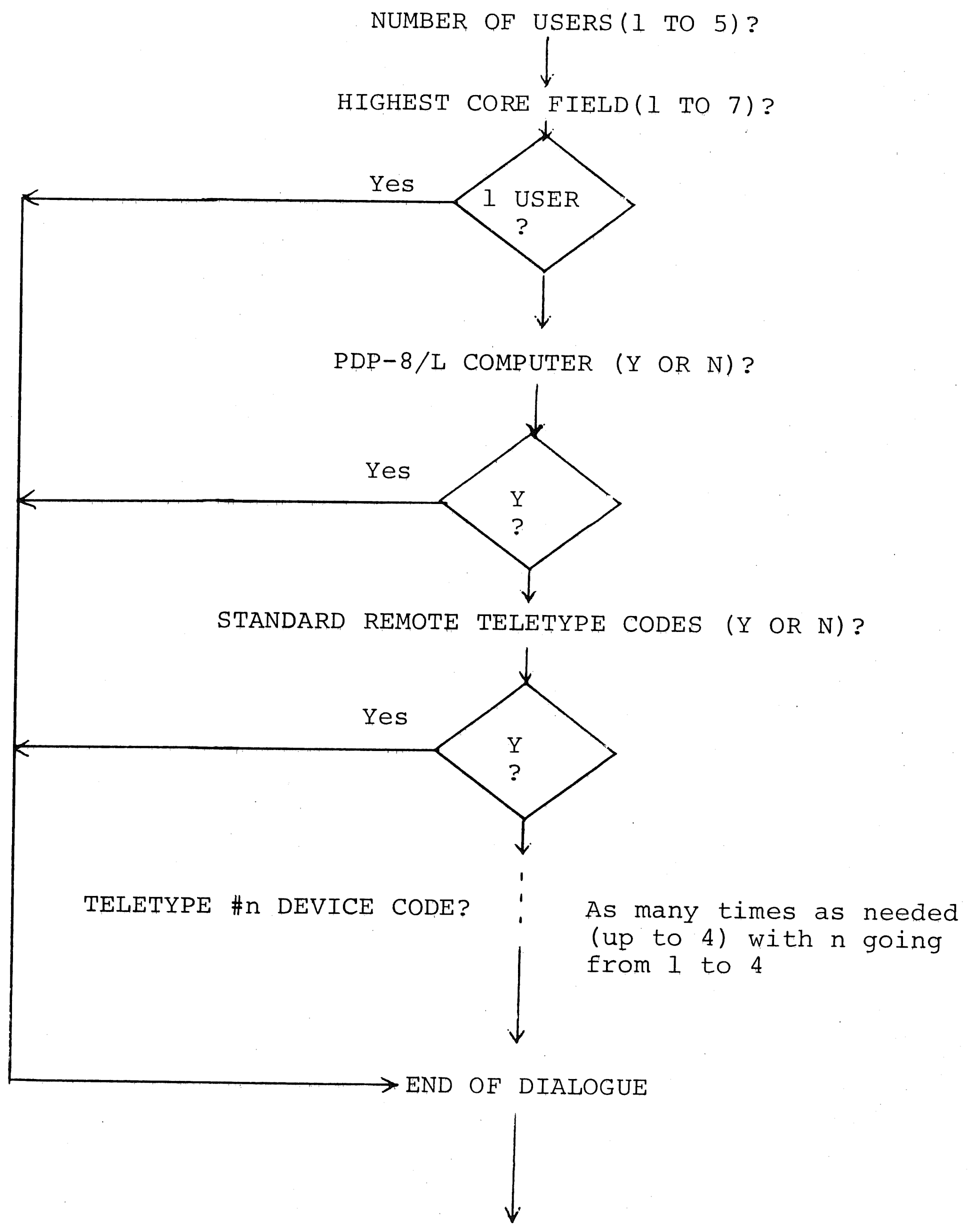

NUMBER OF USERS(1 TO 5)?
HIGHEST CORE FIELD(1 TO 7)?
Yes
1 USER ?
PDP-8/L COMPUTER (Y OR N)?
Yes
Y ?
STANDARD REMOTE TELETYPE CODES (Y OR N)?
Yes
Y ?
TELETYPE #n DEVICE CODE?
As many times as needed (up to 4) with n going from 1 to 4
END OF DIALOGUE

Figure 9-1. Flowchart of Initial Dialogue

BASIC can use a maximum of 28K core memory. Therefore, BASIC will accept a response of 7 but will use only 28K of core memory. More core provides for larger user programs. If you specified the number of users as 1, the dialogue is complete. However, for more than one user, e.g., if a total of four terminals were to be used (the console Teletype plus three remote Teletypes) and your response was 4, BASIC will automatically print the next question immediately after your response. The next question is shown below.

```
EDUSYSTEM 2Ø  BASIC

NUMBER OF USERS(1 TO 5)?4
HIGHEST CORE FIELD(1 TO 7)?1
PDP-8/L COMPUTER(Y OR N)?N
```

Here your response depends on whether you are using a PDP-8/L computer. Type Y (yes) or N (no) accordingly. BASIC will conclude the dialogue when your response is Y. For example, the entire printout might appear as follows:

```
EDUSYSTEM 2Ø  BASIC

NUMBER OF USERS(1 TO 5)?4
HIGHEST CORE FIELD(1 TO 7)?1
PDP-8/L COMPUTER(Y OR N)?Y

END OF DIALOGUE

READY
```

However, when PDP-8/L COMPUTER (Y OR N)? is answered with N, BASIC asks that you respond with a PT08 device code for each terminal to be used (excluding the console terminal). For example, the entire printout might appear as follows:

```
EDUSYSTEM 2Ø  BASIC

NUMBER OF USERS(1 TO 5)?4
HIGHEST CORE FIELD(1 TO 7)?1
PDP-8/L COMPUTER(Y OR N)?N
STANDARD REMOTE TELETYPE CODES(Y OR N)?4
INVALID RESPONSE
STANDARD REMOTE TELETYPE CODES(Y OR N)?N
TELETYPE #1 DEVICE CODE?4Ø
TELETYPE #2 DEVICE CODE?42
TELETYPE #3 DEVICE CODE?44

END OF DIALOGUE

READY
```

For a four-user system with standard PT08 device codes, the responses would be 40, 42, and 44.

NOTE

Standard PT08 device codes are 40, 42, 44, and 46. When a system using PT08 Interface Units is first installed, determine the specific device code for each Teletype and label each Teletype with its specific device code.

Your response to BASIC's first question determines how many times PT08 DEVICE CODE? is printed -- once for each user terminal excluding the console terminal.

When an invalid response is made to any of BASIC's questions, an error message is printed and the question is repeated. For example:

```
STANDARD REMOTE TELETYPE CODES(Y OR N)?4
INVALID RESPONSE
STANDARD REMOTE TELETYPE CODES(Y OR N)?
```

When BASIC has accepted your responses to the initial dialogue, it prints END OF DIALOGUE and then prints READY on each Teletype associated with the specified device codes. BASIC is now ready and waiting for you to type a user program or to issue one of its commands.

9.3 RECONFIGURATION

The Configurator Tape is used to change the number of users, size of core, etc., without completely reloading the System Tape.

To use the Configurator Tape, the system must be inactive, i.e., BASIC must not be running a program and no user typing. The S key may be typed to stop a running program or the listing of a program. To ensure that no one starts typing, turn all Teletypes to OFF. When the system is inactive, load the Configurator Tape as explained in Section 9.1.

NOTE

If some other program has been loaded into memory since the last use of BASIC, you must reload the System Tape as explained in Section 9.1.

9.4 SYSTEM SHUTDOWN

To shut the system down, for overnight or any reason, the procedure in Section 9.3 must be performed to ensure that the system is in the idle state (inactive). Then depress the console STOP switch and turn off the console power.

To restart BASIC, perform the following:

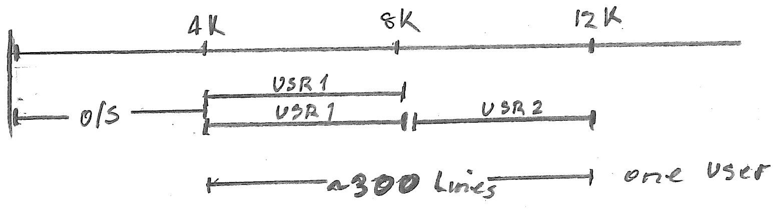
1. Set the console Data Field and Instruction Field to zero.
2. Set the Switch Register to 0200 (BASIC starting address).
3. Press LOAD ADDress and then START.

4. Turn appropriate Teletypes to LINE.
5. BASIC is running and ready for user input.

In the event of a power failure or accidental stopping of the computer, a complete reloading of the System Tape is recommended. Under certain circumstances, restarting at BASIC's starting address (SR=0200) may be successful.

9.5 NUMBER OF LINES ALLOWED

The number of lines allowed varies depending upon the number of users and the amount of core. For example, one user with 8K of core, is allowed approximately 300 lines of average BASIC program statements. Five users with 8K of core are allowed approximately 50 lines each; and the lines allowed for two, three, or four users will fall proportionally between 300 and 50. One user with an 8K system is allowed an equivalent number of lines to two users with a 12K system; two users with an 8K system are allowed an equivalent number of lines to four users with a 12K system, etc.



APPENDIX A

SUMMARY OF BASIC STATEMENTS

Command	Abbreviation	Example of Form	Explanation
LET		LET v=f	Assign the value of the formula f to the variable v
READ	REA	READ v1,v2,..., vn	Variables v1 through vn are assigned the value of the corresponding numbers in the DATA string.
DATA	DAT	DATA n1, n2,..., nn	Numbers n1 through nn are to be associated with corresponding variables in a READ statement.
PRINT	PRI	PRINT a1, a2,..., an	Print out the values of the specified arguments which may be variables, text, or format control characters (, or ;).
GOTO	GOT	GOTO n	Transfer control to line n and continue execution from there.
IF-THEN	IF-THE	IF f1 r f2 THEN n	If the relationship r between the formulas f1 and f2 is true, then transfer control to line n; if not, continue in regular sequence.
IF-GOTO	IF-GOT	IF f1 r f2 GOTO n	Same as IF-THEN.
FOR-TO-STEP	FOR-TO-STE	FOR v=f1 TO f2 STEP f3	Used to implement loops: the variable v is set equal to the formula f1. From this point the loop cycle is completed following which v is incremented after each cycle by f3 until its value is greater than or equal to f2. If STEP f3 is omitted, f3 is assumed to be +1.
NEXT	NEX	NEXT v	Used to tell the computer to return to the FOR statement and execute the loop again until v is greater than or equal to f2.
GOSUB	GOS	GOSUB n	Allows the user to enter a subroutine at several points in the program. Control transfers to line n.
RETURN	RET	RETURN	Must be at the end of each subroutine to enable control to be transferred to the statement following the last GOSUB.
RANDOMIZE	RAN RANDOM	RANDOMIZE	Enables the user to obtain an unreproducible random number sequence in a program using the RND function.

Command	Abbreviation	Example of Form	Explanation
INPUT	INP	INPUT v1, v2, ..., vn	Causes printout of a ? to the user, waits for the user to supply the values of the variables v1 through vn.
REMARK	REM	REMARK text	When typed as the first three letters of a line, allows typing of remarks within a program.
RESTORE	RES	RESTORE	Sets pointer back to the beginning of the string of DATA values.
DEF		DEF FNG (x)= f(x) DEF FNB(x,y)= f(x,y)	The user may define his own functions to be called within his program by putting a DEF statement at the beginning of a program. The function name begins with FN and must have three letters. The function is then equated to a formula f(x) which must be only one line long. Multiple variable function definitions are allowed.
STOP	STO	STOP	Equivalent to transferring control to the END statement
END		END	Last statement in every program; signals completion of the program.
ON-GOTO	ON-GOT	ON f GOTO n,n, ...,n	The formula is evaluated and the first, second, third, etc., line number is selected depending on whether the truncated evaluation is 1, 2, 3, etc.

Functions

In addition to the usual arithmetic operations of addition (+), subtraction (-), multiplication (*), division (/), and exponentiation (↑), BASIC provides the following function capabilities:

SIN(X)	Sine of X	INT(X)	Greatest integer in X
COS(X)	Cosine of X	RND(X)	Random number between 0 and 1,is a repeatable sequence; value of X ignored.
TAN(X)	Tangent of X		
ATN(X)	Arctangent of X		
EXP(X)	e^x (e=2.718282)	SGN(X)	Assign value of +1 if X is positive, 0 if 0, or -1 if negative.
LOG(X)	Log of X (natural logarithm		
ABS(X)	Absolute value of X (|X|)	TAB(X)	Controls the position of the printing head on the Teletype
SQR(X)	Square root of X ($\sqrt{X}$)		
		FIX(X)	Truncate X
		CHR$(X)	use Dec Equiv of ASCII Code

NOTE: Trig functions use radians.

APPENDIX B

SUMMARY OF BASIC EDIT AND CONTROL COMMANDS

Command	Abbreviation	Explanation
BYE	BYE	Clears out entire program.
DELETE	DEL n n DEL n,m	Delete the line with line number n, an alternate form is to type the line number and the RETURN key. Delete the lines with line numbers n through m inclusive.
EDIT	EDI n	Allows the user to search line n for the character typed.
KEY	KEY	Return to KEY (normal) mode.
LIST	LIS LIS n LIS n,m	List the entire program in core. List line n. List lines n through m inclusive.
NEW	NEW	BASIC will clear core.
RUN	RUN	Compile and run the program currently in core.
SCRATCH	SCR	Erase the current program from core.
TAPE	TAP	Enter TAPE mode, characters typed will not echo on the console paper.
S	S	Restarts the Initial Dialogue when used after a wrong response. OR Stops a running program, prints STOP, and then READY.

APPENDIX C

ERROR MESSAGES

WHAT?	Command not understood - ready mode
ERROR 1	Log of negative or zero number requested
ERROR 2	Square root of negative number requested
ERROR 3	Divison by zero requested
ERROR 4	Overflow - exponent greater than approximately +38
ERROR 5	Underflow - exponent less than approximately -38
ERROR 6	Line too long or program too big
ERROR 7	Characters are being typed in too fast - use TAPE command for reading paper tapes
ERROR 8	System overload caused character to be lost
ERROR 9	Program too complex or too many variables
ERROR 10	Missing or illegal operand or double operators
ERROR 11	Missing operator before a left parentheses
ERROR 12	Missing or illegal number
ERROR 13	Too many digits in number
ERROR 14	No DEF for function call
ERROR 15	Missing or mismatched parentheses or illegal dummy variable in DEF
ERROR 16	Wrong number of arguments in DEF call
ERROR 17	Illegal character in DEF expression
ERROR 18	Missing or illegal variable
ERROR 19	Single and double subscripted variables with the same name
ERROR 20	Subscript out of range
ERROR 21	No left parentheses in function
ERROR 22	Illegal user defined function - not FN followed by a letter and a left parentheses
ERROR 23	Mismatched parentheses or missing operator after right parentheses
ERROR 24	Syntax in GOTO
ERROR 25	Syntax in RESTORE
ERROR 26	Syntax in GOSUB
ERROR 27	Syntax in ON
ERROR 28	Index out of range in ON
ERROR 29	Syntax in RETURN
ERROR 30	RETURN without GOSUB

ERROR 31	Missing left parentheses in TAB function
ERROR 32	Syntax in PRINT
ERROR 33	No END statement or END is not the last statement
ERROR 34	Missing or illegal line number
ERROR 35	Attempt to GOTO or GOSUB to a non-existent line
ERROR 36	Missing or illegal relation in IF
ERROR 37	Syntax in IF
ERROR 38	Missing equal sign or improper variable left of the equal sign in LET or FOR
ERROR 39	Subscripted index in FOR
ERROR 40	Syntax in FOR
ERROR 41	No NEXT for FOR
ERROR 42	Syntax in LET
ERROR 43	Syntax in NEXT
ERROR 44	NEXT without FOR
ERROR 45	Too much data typed in or illegal character in DATA or the data typed in
ERROR 46	Illegal character or function in INPUT or READ
ERROR 47	Out of data
ERROR 48	Unrecognized command - RUN mode

APPENDIX D

IMPLEMENTATION NOTES

The BASIC language is compatible with Dartmouth BASIC except as noted below:

1. There are no matrix operations.

2. There are no character string instructions.

3. There are no facilities which allow reading or writing of programs or data files on disk.

4. User defined functions are restricted to one line.

APPENDIX E

READ-IN MODE LOADER

The Read-In Mode (RIM) Loader is the very first program loaded into the computer; it is loaded by toggling 17 instructions into core using the console switches. The RIM Loader instructs the computer to receive and store, in core, data punched on paper tape in RIM coded format -- primarily the Binary Loader.

There are two RIM Loader programs: one is used when the input is to be from the low-speed (Teletype) paper tape reader, and the other is used when input is to be from the high-speed paper tape reader. The locations and corresponding instructions for both programs are listed in Table E-1. The procedure for loading (toggling) the RIM program into core is illustrated in Figure E-1. The RIM Loader is loaded into field zero of core.

Table E-1. RIM Loader PRograms

	Instruction	
Location	Low-Speed	High-Speed
7756	6032	6014
7757	6031	6011
7760	5357	5357
7761	6036	6016
7762	7106	7106
7763	7006	7006
7764	7510	7510
7765	5357	5374
7766	7006	7006
7767	6031	6011
7770	5367	5367
7771	6034	6016
7772	7420	7420
7773	3776	3776
7774	3376	3376
7775	5356	5357
7776	0000	0000

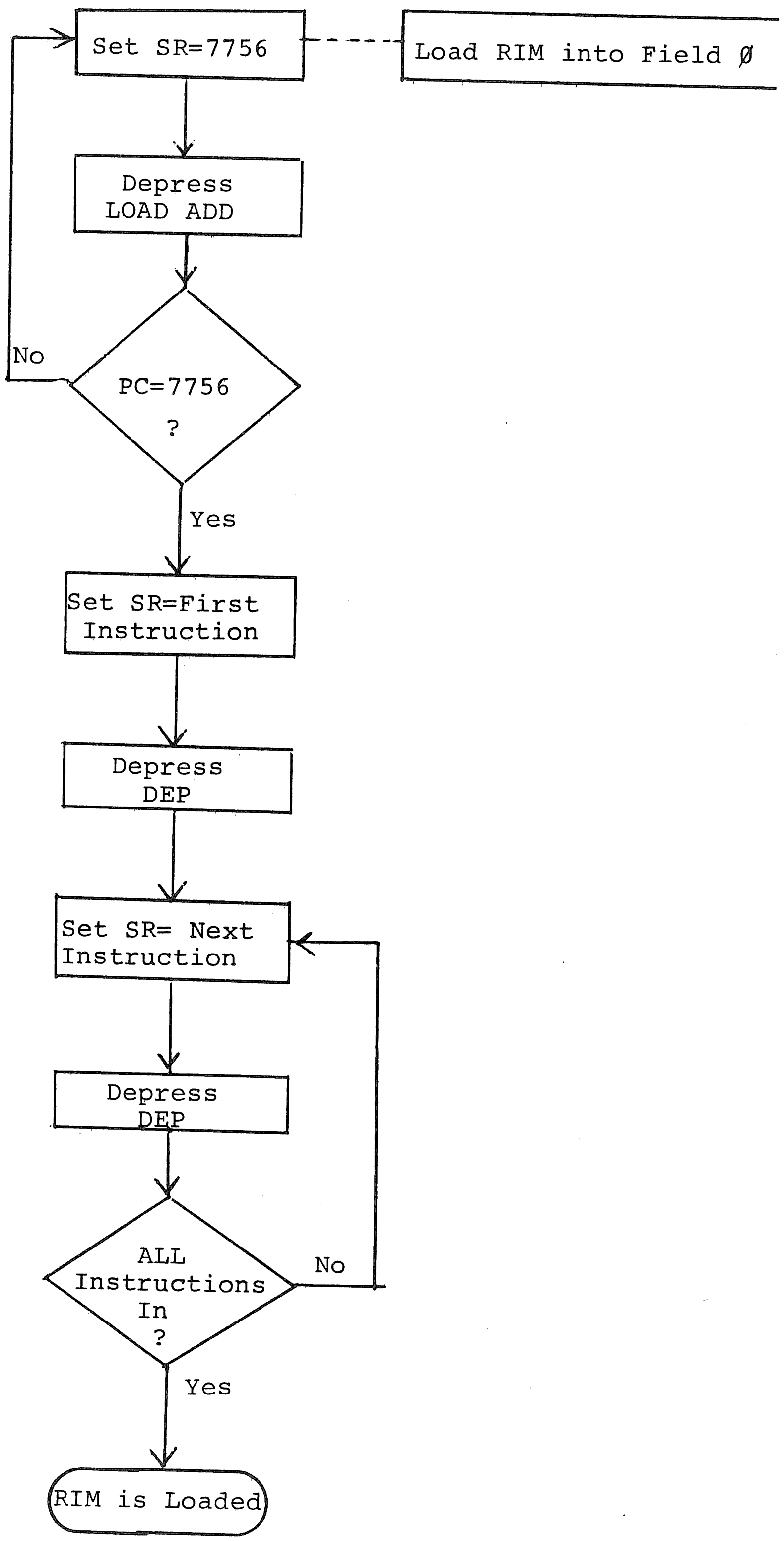


Figure E-1. Loading the RIM Loader

After RIM has been loaded, it is good programming practice to verify that all instructions were stored properly. This can be done by performing the steps illustrated in Figure E-2, which also shows how to correct an incorrectly stored instruction.

When loaded, the RIM Loader occupies absolute locations 7756 through 7776. The BASIC System does not use the RIM locations; therefore, RIM need not be reloaded to load the Configurator Tape after running the System unless, of course, the contents of the RIM locations have been altered by the user.

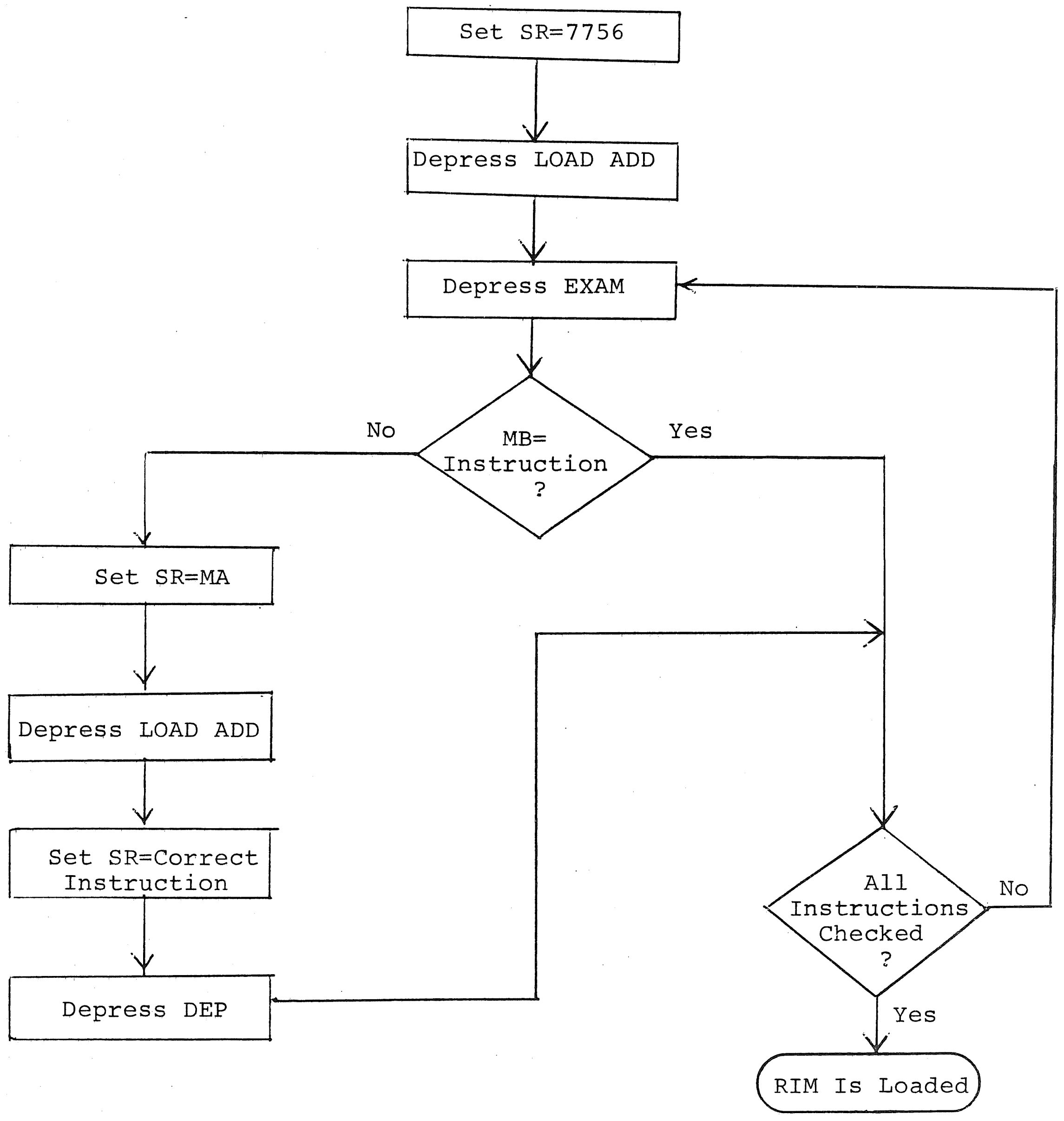


Figure E-2. Checking the RIM Loader

APPENDIX F

for CHR$(x)

ASCII CHARACTER SET

Character	Octal Code	Dec Code
A	301	65
B	302	66
C	303	67
D	304	68
E	305	69
F	306	70
G	307	71
H	310	72
I	311	73
J	312	74
K	313	75
L	314	76
M	315	77
N	316	78
O	317	79
P	320	80
Q	321	81
R	322	82
S	323	83
T	324	84
U	325	85
V	326	86
W	327	87
X	330	88
Y	331	89
Z	332	90
@	300	64
0	260	48
1	261	49
2	262	50
3	263	51
4	264	52
5	265	53
6	266	54
7	267	55
8	270	56
9	271	57
!	241	33
"	242	34
#	243	35
$	244	36
%	245	37
&	246	38
'	247	39
(	250	40
)	251	41
*	252	42
+	253	43
,	254	44
-	255	45
.	256	46
/	257	47
:	272	58
;	273	59
=	275	61
?	277	63
[	333	91
]	335	93
Bell	207	07
Tab	211	09
Line Feed	212	10
Carriage-Return	215	13
Space	240	32
Rubout	377	127
←	337	127
Form	214	12

INDEX

HOW TO OBTAIN SOFTWARE INFORMATION

Announcements of new and revised software, as well as programming notes, software problems, and documentation corrections are published by Software Information Service in the following newsletters:

Digital Software News for the PDP-8 and PDP-12

Digital Software News for the PDP-9/15 Family

Digital Software News for the PDP-11

These newsletters contain information to update the cumulative

Software Performance Summary for the PDP-8 and PDP-12

Software Performance Summary for the PDP-9/15 Family

Software Performance Summary for the PDP-11

The appropriate edition of the Software Performance Summary is included in each basic software kit for new customers. Additional copies may be requested without charge.

Any questions or problems on the articles contained in these publications or concerning the use of Digital's software should be reported to the Software Specialist or Sales Engineer at the nearest Digital office.

New and revised software and manuals, and current issues of the Software Performance Summary are available from the Program Library. To place an order, write to:

Program Library
Digital Equipment Corporation
146 Main Street, Building 1-2
Maynard, Massachusetts 01754

When ordering, include the code number and a brief description of the program or manual requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of available programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information, please write to:

DECUS
Digital Equipment Corporation
146 Main Street, Building 3-5
Maynard, Massachusetts 01754

EduSystem-20
User's Guide
DEC-8I-KJXA-D

READER'S COMMENTS

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

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

Did you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Please state your position. ______ Date: ______

Name: ______ Organization: ______

Street: ______ Department: ______

City: ______ State: ______ Zip or Country ______

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

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

Postage will be paid by:

digital

Digital Equipment Corporation
Software Information Services
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754