U. S. ARMY

ENGINEER

SCHOOL

BASIC

COMPUTER PROGRAMING LANGUAGE MANUAL

INSTRUCTIONAL DATA SYSTEMS PROJECT OFFICE

FORT BELVOIR, VIRGINIA

PREFACE

The development of the BASIC language was supported by the National Science Foundation under the terms of a grant to Dartmouth College. The project, under the direction of Professors J. G. Kemeny and T. E. Kurtz, included creation of a compiler for the BASIC language and the necessary executiveroutines for the GE 235, Datanet 30 Computer Systems. Reference material has been used from "A Manual for BASIC", the elementary algebraic language designed for use with the Dartmouth Time Sharing System. Dartmouth College permitted this reproduction for non-commercial use.

C O N T E N T S

C O N T E N T S Continued

INDEX

BASIC MANUAL TOPIC PAGE # GUIDE

TOPIC	PAGE
Arrays	11
Arithmetic Symbols	2
Comma Control	7
Computer Connection	24
DATA Statement	5
De-bugging	20
DIM Statement	11
END Statement	7
Errors	20
Error Messages	
265 System	31
420 System	35
Flow Charting	14
FOR and NEXT Statements	10,18
Functions	3
GOTO Statements	6
IF-THEN Statement	9
LET Statement	5
Line Numbers	4,21,22
Loops	10,13,18,20
Numbers	2
Paper Tape	26
PRINT Statement	7
Print Zones	7
READ Statement	6
Relations	4,9
REM Statement	4
Semi-colon Control	7
SOP	41
Summary of Statements	4
System Commands	29,30
TTY Keyboard	25
TTY Keys	22
Variables	
Non-subscripted	2
Subscripted	2,11

INTRODUCTION AND PURPOSE

The Engineer School is currently in the process of obtaining an academic computer to support its training mission. Since the faculty will be using this computer to support their instruction, this course is necessary for two reasons. It is obvious that the faculty will have to know how to use the computer when it is installed; more fundamentally, however, the instructors are needed to develop applications and write programs to enable the Instructional Data Systems Project Office to write specifications for a system that will best fulfill our needs.

The purpose of this course is to introduce you to the BASIC programming language and to provide you with a few computer programming techniques. BASIC is a programming language that was developed at Dartmouth College to overcome the problem of slow turnaround particular in an academic environment. BASIC is an easy programming language to learn and is relatively powerful. Yet, the simplicity of BASIC allows one to write his own programs after a minimum of instruction.

To be successful computer programmers, you must meet two basic requirements: (1) you must be intimately familiar with the problem you want solved, and (2) you must be able to effectively communicate with the computer. The techniques and language learned in this course are all that you will need to solve the most difficult problems. This supplement and your BASIC reference manual will provide you with most of the details you will need to know in order to write the more difficult programs. This course will fill in most of the gaps, EXPERIENCE will complete your education in programming.

2. BASIC COMPONENTS

NUMBERS

A number may contain up to 9 digits exclusive of the sign or decimal point. All numbers in the BASIC system are double-precision floating point numbers. The numbers and constants that you will be using in most of your calculations will be expressed normally as you do now (e.g., - 1234.56, 8.432, -85, 607). However, the range of numbers may be extended by expressing the number as a power of ten. Using the letter "E" to stand for "times ten to the power," (exponential), you can write numbers in this form between E-77 and E77 inclusive. Positive "E" Notation numbers do not require a plus sign (e.g., use E77 not E+77). The format for this number is:

$\pm$ XXXXXXXXX E ($\pm$) YY

Example:- 12345E -36 or 6789E36. 1ØØ may be written 1ØE1 or 1E2 but not E2.

NOTE: Any unsigned number is assumed to be positive.

VARIABLES

There are two types of variables in BASIC: (1) A subscripted variable and, (2) non-subscripted variable. A subscripted variable is expressed by any single alphabetic character followed by a subscript in parentheses. (The subscript itself can be a variable.) For example: A(6), B(6,7), C(A*B), D(I) are all subscripted variables. A non-subscripted variable is expressed as any single alphabetic character or any single alphabetic character followed by a single digit. For example: A, B6, C, C1, are all acceptable non-subscripted variables.

EXPRESSIONS

Expressions are formed by combining variables and numbers together with arithmetic operations and parentheses just as in ordinary mathematical formulas. The ARITHMETIC SYMBOLS listed in order of priority* of execution are:

	SYMBOL	EXAMPLE	MEANING
Parentheses	( )	(A+B)	Use for clarity
Exponential	↑	X↑2	Find X^2
Multiplication	*	A*B	Multi B by A
Division	/	A/B	Divide A by B
Addition	+	A+B	Add B to A
Subtraction	-	A-B	Sub B from A

*NOTE: The order of priorities for computing is according to the following rules:

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

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

3. In the absence of parentheses in a formula that includes several multiply-divides, or several addition-subtractions, the computer works from left to right.

If there is ever any question in your mind about the priority, put in more parentheses to avoid possible ambiguities.

Examples:

How Written	How Preformed	How Meant
A/B/C	(A/B) /C	A/(B/C)
A-B-C	(A-B) -C	A-(B-C)
A↑B↑C	(A↑B) ↑ C	A↑(B↑C)
A*B/C*D↑H	(A*B)/C *(D↑H)	(A*B)/ (C*D)↑ H

FUNCTIONS

In addition to the arithmetic operations, some of the more common standard functions are available. These BASIC functions are:

FUNCTION NAME	PURPOSE	
SIN(X)	SINE of X)	
COS(X)	COSINE of X)	
TAN(X)	TANGENT of X)	X must be expressed
ATN(X)	ARCTANGENT of X)	in radians
EXP(X)	Natural exponential of X, e.g., e^X	
ABS(X)	Absolute value of X, e.g., 1X1	
LOG(X)	Natural logrithm of X, e.g., ln(X)	
SQR(X)	Square root of X, e.g., $\sqrt{X}$	
RND(X)	Produces random numbers between 0 and 1	
INT(X)	Chooses integer part of X	

The DEF STATEMENT permits the user to define a function other than the standard functions listed above. See Advanced BASIC Techniques.

RELATIONS

These six mathematical symbols are used in IF-THEN statements where it is necessary to compare values.

SYMBOL	EXAMPLE	MEANING
<	A < B	Less Than
=	A = B	Equal To
<=	A <= B	Less than or equal to
>	A > B	Greater than
>=	A >= B	Greater than or equal to
<>	A <> B	Less than or greater than (Not Equal)

3. SUMMARY OF BASIC STATEMENTS

BASIC STATEMENTS

This is a review of each BASIC statement, its function, how it is used and when it is used in programming. Each BASIC statement is made up of three components in the following format:

[Line Number] KEYWORD [Labels and/or Expressions]

The Notation [.....] is used in the general format of each statement to denote a particular unspecified quantity. The [ ] (Brackets) are not actually used in the statements as you will note from the examples.

REM

The REM statement allows you to insert explanatory remarks within a program. The computer completely ignores every part of the line following REM, which allows you to include directions for using the program, identifications of the parts of a long program, or anything else that you may need. The format of the REM statement is:

[Line Numbers] REM [any comment]

Examples of REM STATEMENTS:

```
1ØØ REM insert DATA in lines 5ØØ-53Ø.

9ØØ REM This sub-routine solves equations.

5 REM Find integer part of X.
```

LET

Probably the single most important statement in BASIC is the LET statement. It is a command to the computer to perform a certain operation to the right of the "equals" and assign the results to a specific variable on the left. The format of the LET STATEMENT is:

[Line Number] LET [Subscripted or non-subscripted variable] = [Expression]

Examples of LET STATEMENTS:

```
1ØØ LET X1 = Ø
```

This says: Place zero in memory location X1.

```
5ØØ LET X2 = X2 + SQR(R*R+S*S)
```

This says: Find the square root of the sum of R*R + S*S, add this number to the previous number in memory location X2 and place the new number in X2.

```
299 LET A4 = (S1 + A3 - Q)↑2
```

This says: Add A3 to S1, subtract from this number, the number represented by Q, raise the resulting number to the 2nd power and, finally, place the results in the memory position A4.

DATA

The format of the DATA STATEMENT is:

[Line Number] DATA [List of numbers separated by commas]

Examples of DATA STATEMENTS:

```
9ØØ DATA 14,67.2

9Ø1 DATA 642,86,4.97E-7,62
```

These statements say: Take the numbers following the name DATA and place them in a table before the program is run. If the above two DATA statements were in the same program, the data table in the computer would contain all the numbers beginning with those following the DATA statement with the lowest statement number. Each DATA statement can contain as many numbers as can be typed on a single line. Each new line of data must have a statement number and contain the name DATA.

READ

The format of the READ STATEMENT is:

[Line Number] READ [List of Variables]

Examples of READ STATEMENTS:

```
1Ø READ A,B,C
```

This says: Take the first three numbers from the DATA table and place them in memory locations A, B and C, respectively. If this statement is encountered at another time during the problem, the next three numbers would be selected as A, B and C, and moved to their locations in memory.

```
1Ø READ A,B,C
15 READ X1,X2,Q,R
```

These statements say: Take the first three numbers from the DATA table and place them in memory locations A, B, and C, and take the next four numbers and place them in memory locations X1,X2,Q and R. If the end of the DATA table is reached before any READ statement is satisfied, the computer will stop and print "OUT OF DATA".

GOTO

The format of the GOTO STATEMENT is:

[Line Number] GOTO [Line Number]

Examples of GOTO STATEMENTS:

```
9ØØ GOTO 1Ø
```

This says: Go to statement number 1Ø for the next instruction.

```
25 GOTO 942
```

This says: Go to statement number 942 for the next instruction. The GOTO statement is an unconditional transfer statement. Each time it is encountered in the program, the computer goes to the line number following the word GOTO for the next statement.

*NOTE: A variable cannot be used for the line number.

END

The format of the END STATEMENT is:

[Line Number] END

Example of END STATEMENT:

```
999999 END
```

An END statement is required in all programs. It must also be the statement with the highest line number in the program.

PRINT

The PRINT statement is extremely useful and versatile. A number of common uses are:

a. To print the result of computations.

b. To print out verbatim a label included in the program.

c. To print out a combination of A and B.

d. To skip a line in the program.

Print Zones - The PRINT line is made up of 72 characters which are divided into zones; the first 4 zones are 15 characters long, the last one has 12 characters.

In order to control the format of the printed output, we have the option of using either "comma" or "semi-colon" control.

Comma Control - The use of a comma is a signal to move to the next print zone (15 characters long) or, if the fifth print zone has just been filled, to move to the first print zone of the next line.

Semi-Colon Control - The use of a semi-colon is a signal to move to the next multiple of three characters for the next answer. Thus, by using a semi-colon, a programmer can pack 11 three digit numbers per line, 8 six digit numbers per line, or 6 nine digit numbers per line as opposed to 5 numbers when using a comma.

The format of the PRINT STATEMENT is:

[Line Number] PRINT [Labels and/or Expressions]

Examples of type a:

```
    1ØØ PRINT X,Y,Z,Q↑2,SQR(X3)

    12Ø PRINT A,B,C,B*B-4*A*C,EXP(A-B)
```

Statement 1ØØ will cause the number X to be printed in the first print zone, Y in the next and Z in the third; it will evaluate Q↑2 and print this number in the fourth zone and it will evaluate SQR(X3) and place the resulting number in the fifth zone. The expressions to be computed can be of any complexity. The above examples demonstrate the ability to print out the values of variables that have been computed elsewhere in our program. It also demonstrates the ability to compute and print out values from self-contained expressions.

Examples of type b:

```
    1ØØ PRINT "COMPLEX ROOTS"

    115 PRINT 1,2,3,4,5

    13Ø PRINT "NUMBER", "VALUE", "TOTAL"
```

The above statements demonstrate how alphabetic messages or labels may be printed out verbatim when they are enclosed within quotation marks; numeric labels may be printed directly without using quotation marks; and by using Comma Control, heading type labels may be printed in the different zones.

Examples of type c:

```
    1ØØ PRINT "THE VALUE OF Y IS" Y

    11Ø PRINT I, "SQUAREROOT =" SQR(I)

    12Ø PRINT "THE SUM OF THE FIRST "N" INTEGERS IS" S
```

The combination of labels or messages mixed with variables and expressions provides the programmer with the most useful output format available. We find that clarity and understanding of your solutions will be greatly improved with this type of print statement. Statement 12Ø will look like this: The sum of the first 10 integers is 55. (For N = 10) Even if some period of time has elapsed between the running and actual use of this program's output, the solutions are still readily understandable.

Example of type d:

```
100 PRINT
```

This statement causes the computer to print "Nothing" on a line (i.e. skip a line). There is no utility in using this statement unless you are having the computer list your program. If this is the case, the readability will be greatly improved.

Much more variety is permitted in PRINT statements than we have shown here. The additional flexibility is explained in the Advance BASIC Section.

IF-THEN

In a programming sequence, it is often necessary to alter the course of a program depending upon the outcome of a particular operation. This can be accomplished with the conditional transfer statement: the IF-THEN statement. The format of this statement is:

[Line Number] IF [Expression] [Relation] [Expression] THEN [Line Number]

The six acceptable relations are:

<	Less than
=	Equal to
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<>	Not equal

Examples of IF-THEN statements:

```
 50 IF A = B THEN 90
100 IF (R*R-S*S)<= 0 THEN 50
110 IF (X- (Y↑I)*Z)>(C/R)↑N THEN 80
```

Statement 50 says: If the condition of the relationship (A =B) is met, THEN transfer to Statement 90. If the condition is not met, THEN continue with the next statement after 50.

Statement 1ØØ says: If the Expression (R^2 - S^2) is less than or equal to zero, THEN go to Statement 5Ø for the next instruction. If the condition is greater than zero, the computer continues on to the next statement.

*NOTE: If the condition in the first statement is met, the computer transfers ahead, and in the second statement the transfer is a loop back.

Statement 11Ø compares two expressions for a greater than relationship. If this condition is met, the computer loops back to Statement 8Ø.

FOR AND NEXT STATEMENTS

The FOR and NEXT statements are especially useful in Loop Control. The formats of these statements are:

```
 ┌→[Line Number]  FOR  [Subscripted or non-    ]  =  [Expression]  TO
 │                      [subscripted variable   ]
 │
 Loop      [Expression]  STEP  [Expression]
 │
 └─[Line Number]  NEXT  [Subscripted or non-    ]
                         [subscripted variable   ]
```

Examples of FOR and NEXT STATEMENTS:

Example 1

```
     5 READ Y,N,A
 ┌→ 1Ø FOR I = 1 To N
 │  2Ø LET Y = Y + 1
 Loop
 │  3Ø PRINT Y
 └─ 4Ø NEXT I
    5Ø IF A = Ø THEN 99
```

The FOR Statement says: Let I = 1, then compare it with the value of N. If I is less than or equal to N, continue through the loop with Statement 20.

The NEXT Statement says: Go back to Statement 1Ø (FOR Statement) where I is advanced by 1. (I is advanced by the step expression. If no step is given, it automatically advances by +1.) I's new value is compared to N. If I is greater than N, the computer will go to Statement 5Ø, if I is less than or equal to N, the computer will continue back through the loop.

Example 2

```
  1ØØ FOR X = XØ TO A STEP X1

  11Ø LET Q = 3.1416*(X-XØ)↑2
Loop
  12Ø PRINT X,Q

  13Ø NEXT X
```

These statements say: Let X = XØ, then compare it with the value in A, if X is less than or equal to A (in the case of positive step size) go to statement number 11Ø. The program proceeds to 11Ø, 12Ø and 13Ø, which sends it back to 1ØØ. At Statement 1ØØ, the computer tests the size of X+X1 with A, if it is less than or equal to A, the computer assigns X+X1 to X and proceeds thru the loop again. If it is greater than A, the computer goes to the statement following Statement 13Ø. The value of X will remain as it was the last time through the loop. In the case of negative step size, the FOR Statement goes through the body of the loop if the variable has a value greater than or equal to the final value. If the step is not used, the computer assumes a value of one. Every FOR Statement must have one, and only one, corresponding NEXT Statement.

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

*NOTE: If you write FOR X = 3 to -3 without a negative STEP expression, the loop will not be performed and the computer will proceed to the statement immediately following the corresponding NEXT Statement.

<u>DIM</u>

BASIC provides that each list has a subscript running from 0 to 10, inclusive. If you desire lists or tables larger than this, you must use a dimension (DIM) statement. Format of this statement is:

[Line Number   DIM   [Letter (Integer)] or [Letter (Integer, Integer)]

Examples of the DIM STATEMENT:

```
    1Ø DIM A(17)
```

This says: Reserve 18 spaces for List A running from A(Ø) to A(17).

```
    2Ø DIM A(17),S(15,2Ø)
```

This says: Reserve 18 spaces for List A running from A(Ø) to A(17) and reserve a table for S running from Ø to 15 rows and Ø to 2Ø columns.

Uses of the DIM STATEMENT:

```
    1Ø DIM A(17)

    2Ø FOR X = Ø TO 17

    3Ø READ A(X)

    4Ø NEXT X
```

This says: Statement 1Ø says reserve 18 spaces for List A. Statements 2Ø, 3Ø and 4Ø will cause the computer to READ 18 numbers into List A.

```
    1Ø DIM S(15,2Ø)

    2Ø FOR X = Ø TO 15

    3Ø FOR Y = Ø TO 2Ø

    4Ø READ S(X,Y)

    5Ø NEXT Y

    6Ø NEXT Y
```

This says: Fill the S table with data beginning with S(Ø,Ø),S(Ø,1),.... S(Ø,2Ø),S(1,Ø),S(1,1)......S(15,2Ø)

<u>STOP, DEF, GOSUB, RETURN, INPUT and MAT</u>

These advanced statements are all adequately explained in the Advanced BASIC Section. Every programmer should be familiar with them and their use. Experience with their use, as well as the use of the special functions, should be gained through the solution of real or practice problems.

4. LOOPING TECHNIQUES

Perhaps the single most important programming technique is that of a "loop". If we must write a new program each time we want to make a slight change, the usefulness of our powerful computer will be substantially limited. To fully utilize the capabilities of the computer, we can prepare portions of our program to be repeated many times with slight changes each time. This "looping" technique may be either implied or direct loop initialization.

Implied Loops - Implied loops may be developed by transferring back to a lower statement number. This transferring is accomplished with either a GOTO Statement or an IF-THEN Statement.

Perhaps, it is best to illustrate and explain these looping techniques with example problems. If we need a table of the first 50 numbers "raised to the power of two" (squared), we can write a program with 51 statements or by using loops, reduce the number of statements to 5 or 8.

Without Loop

```
 1 PRINT 1, 1↑2
 2 PRINT 2, 2↑2
 3 PRINT 3, 3↑2
   .
   .
   .
49 PRINT 49, 49↑2
5Ø PRINT 5Ø, 5Ø↑2
51 END
```

With IF-THEN Loop

```
         1 LET X = Ø
    ┌──> 2 LET X = X + 1
Loop│    3 PRINT X, X↑2
    └─── 4 IF X<5Ø THEN 2
         5 END
```

There are four characteristics that each loop must contain: initialization, modification, body, and an exit test. In the sample loop problem, line 1 initializes the loop, line 2 modifies the value of X by increasing it by one each time through the loop, line 3 provides printed output each time through the loop, and line 4 is an exit test to see if the loop has been repeated 50 times.

This same problem solved with a GOTO loop will look like this:

```
          1 LET X = Ø
          2 LET Y = 5Ø
    ┌───> 3 LET Y = Y-1
    │  ┌──4 IF Y<Ø THEN 8
Loop│  │  5 LET X = X+1
    │  │  6 PRINT X, X↑2
    └──┼──7 GOTO 3
       └>*8 END
```

*NOTE: Exit Test transfers out.

NOTE: Statement 4 is an exit test for this loop. If this statement were to be omitted, we would have an infinite loop (no exit). Therefore, it is very important to include an exit test in every loop you program.

Because loops are so important to many applications of programming, BASIC provides two statements to specify a loop. They are the FOR and NEXT Statements and their use in solving the same problem of 1 to 5Ø squared is illustrated below in just four statements:

```
1 FOR X = 1 TO 5Ø
2 PRINT X, X↑2
3 NEXT X
4 END
```

FOR and NEXT Statements are fully discussed in The Summary of BASIC Statements (Section 3). However, for many applications in programming it is useful to have loops contained within loops. These are called nested loops and may be initiated with FOR and NEXT Statements or IF-THEN and GOTO Statements. Beware, these loops must be actually nested and must never be allowed to cross. Note the illustrations:

<u>Not Permitted</u>

```
         FOR X
Crossed  FOR Y
  Loop   NEXT X
         NEXT Y
```

<u>Permitted</u>

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

The example given on GOTO "looping" demonstrates the ability to transfer out of the body of a loop. However, great care must be used if you need to transfer into the body of a loop. Generally, you will never need to do this.

5. <u>PROGRAM DEVELOPMENT</u>

<u>PROGRAM ASSEMBLY BY FLOW CHARTING</u>

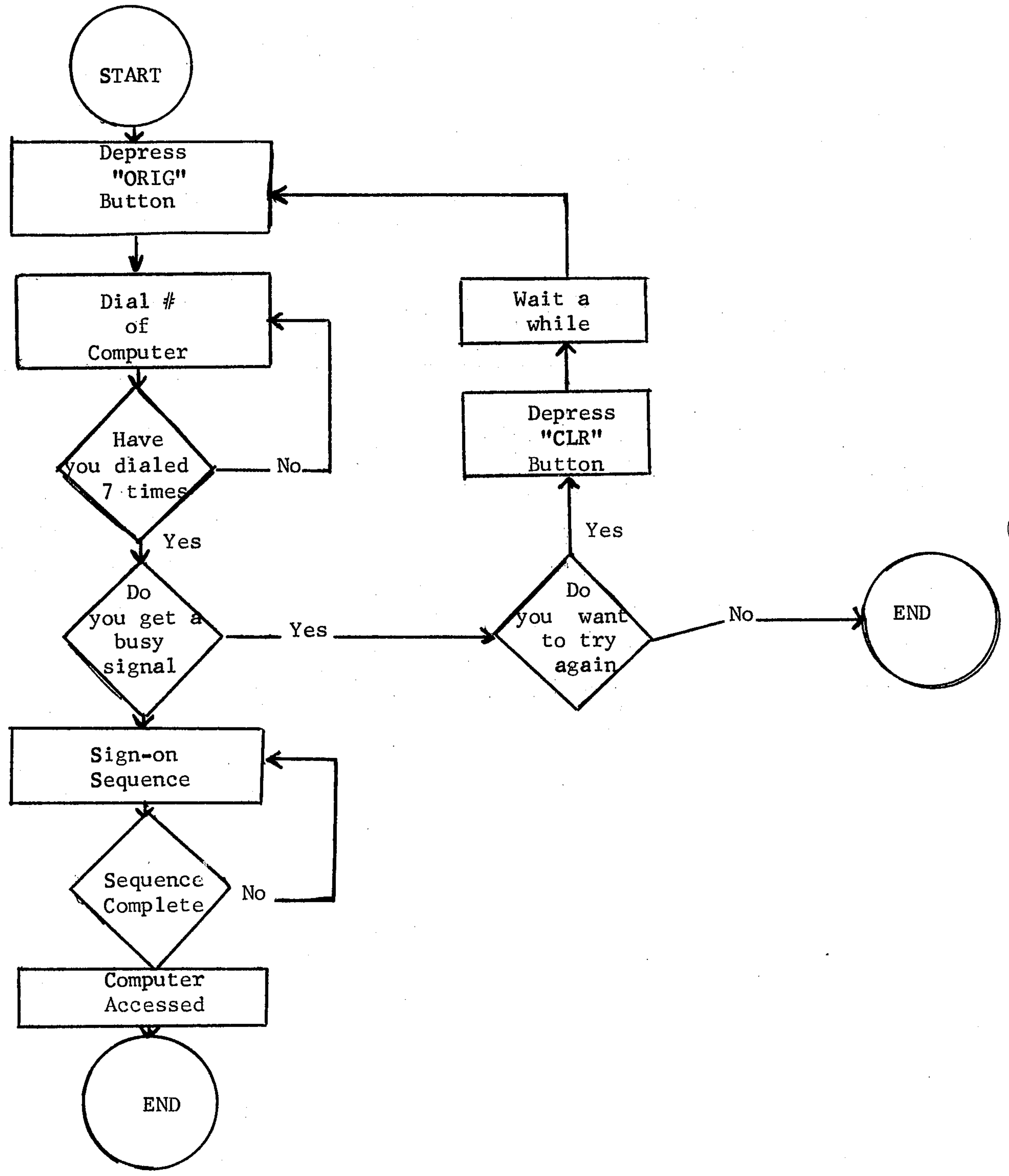
In order to solve your problem through the use of a computer, it is necessary to (1) recognize the problem to be solved, (2) analyze the problem thoroughly, (3) decide on a solution procedure, (4) create a step-by-step logical sequence of instructions which perform this procedure. This step-by-step set of instructions is called a <u>program</u>. The best method of outlining this problem for programming is the <u>Flow Chart</u>. A flow chart is made up of a combination of geometric symbols which represent the various operations required in the solution of your problem. These charts are usually drawn to be read down and to the right.

The illustration on the next page shows how one might describe the various steps involved in using a teletype machine to access a computer. This illustration may appear to include some unnecessary effort for solution of such a simple problem. However, the intent is to demonstrate that the steps must be organized, and to suggest that this method of Flow Charting is very useful, even necessary, when applied to more complex problems.

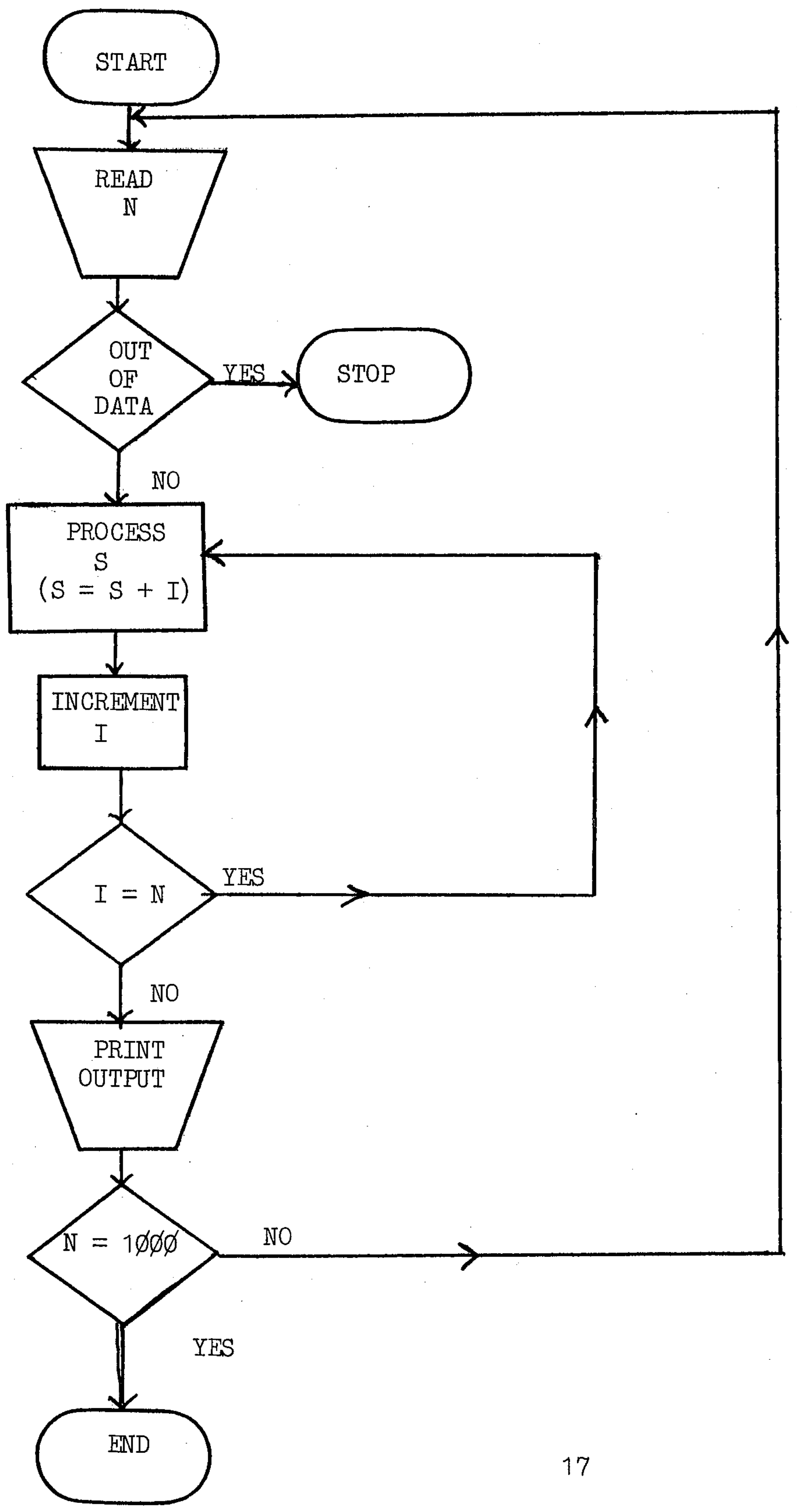
The symbols and techniques needed for flow charting will be discussed in advanced classes. We note it here for your information.

FLOW CHART

The logic in using a teletype to access a computer



PROBLEM: Find the sum of the first N integers.



NOTE:

NESTED LOOPS

EXAMPLE PROGRAM AND REMARKS

PROBLEM: Find the sum of the first N integers.

PROGRAM 1:

```
 1Ø REM Find the sum of the first N integers

 2Ø READ N

 3Ø LET S = Ø

 4Ø FOR I = 1 to N

 5Ø LET S = S + I

 6Ø NEXT I

 7Ø PRINT "THE SUM OF THE FIRST"; N; "INTEGERS IS",S

 8Ø IF N = 1ØØØ THEN 999

 9Ø GOTO 2Ø

1ØØ DATA 1Ø,2Ø,1ØØ,1ØØØ

999 END

 RUN

 THE SUM OF THE FIRST 1Ø INTEGERS IS 55
             .         2Ø               21Ø
             .        1ØØ               5Ø5Ø
             .
             .
 THE SUM OF THE FIRST 1ØØØ INTEGERS IS 5ØØ5ØØ
```

STATEMENT MEANINGS FOR PROGRAM 1

1Ø REM	A remark statement that tells what this program does.
2Ø READ	Causes the first value in the DATA statement to be placed into memory location N.
3Ø LET	Causes zero to be placed in memory location S.
4Ø FOR	This statement says: Let I = 1, then compare it with the value in N, if I is less than or equal to N, go to the next statement.
5Ø LET	Causes the number in location S to be added to the value of I and their sum to be placed back in memory location S.
6Ø NEXT	Sends the computer back to statement 4Ø where I is advanced by 1 and its value is compared to N. If I is greater than N, the computer goes to the statement following 6Ø, if not, back through the loop.
7Ø PRINT	Prints the tag "The Sum of The First '<u>N</u>' integers is '<u>S</u>' and prints the value in memory location N and S in the space provided following the tag.
8Ø IF	Test the magnitude of N. If it is equal to the last value in the DATA statement, the program is completed.
9Ø GOTO	Tells the computer to execute statement number 2Ø because there is uncomputed DATA.
1ØØ DATA	Contains the DATA for this program.
999 END	Tells the computer that this is the last statement.

PROGRAM 2

PROBLEM: Find the sum of the first N integers.

This is the same problem that we solved in program 1. However, we have used implied loops in place of the direct loops in this program. Note this difference and how it may be useful to your programming.

```
1Ø REM FIND THE SUM OF THE FIRST N INTEGERS

2Ø READ N

25 LET I = N

3Ø LET S = Ø

4Ø LET S = S+I

5Ø LET I = I-1

6Ø IF I  Ø THEN 4Ø

7Ø PRINT "THE SUM OF THE FIRST"; N; "INTEGERS IS"; S

75 GOTO 2Ø

8Ø DATA 1Ø,2Ø,1ØØ,1ØØØ

999 END
```

NOTE: The solutions for programs one and two are the same, however, the body of each program is very different. One test for a good program is, "does it produce the correct answer?" Any program that produces the correct answer is a good program. However, efficiency is needed whenever possible for cost reductions.

ERRORS AND DE-BUGGING

If the first run of your new program is error-free and produces the correct solutions, you will be most fortunate. However, it is much more common that you will have errors that will need to be corrected. You will find that there are two types of errors: (1) Errors of form, or grammatical errors, that prevent even the running of your program, and (2) logical errors in the program which produce wrong solutions or even no solution to be printed.

Errors of form, or grammatical errors, often prevent the complete running of your program. Usually error messages will be printed out instead of the expected solutions. These error messages will give the nature of the error, and the line number in which the error occurred. (NOTE: These errors may occur after part of your solutions have been printed.) Various error messages for errors of form, together with their interpretations, are given in Appendix C.

Logical errors are usually much harder to uncover, particularly when the program appears to give nearly correct solutions to your problem.

The whole process of locating Logic errors or "de-bugging" a program can be approached in many different ways. One is use of "Test Cases" where a program is run with data for which the solution is known. However, test data must be carefully selected if you are to succeed in locating all the bugs. Difficult de-bugging problems sometimes cannot be solved by supplying test cases. Another technique called "tracing" will be most useful.

Tracing not only prints out the final solutions, but also the results of the intermediate calculations. This is accomplished by using "print statements" to print the value of a variable immediately after a calculation has been performed. Also, by printing paragraph or section names at critical points in the program will allow you to follow the flows in program logic. By comparing these results with presumably correct ones, you can usually find the errors in your program.

Another technique, which we strongly recommend, is a generous usage of REM Statements. Any program, especially a long complicated one, should be supplied with enough REM Statements to explain adequately how the program works.

Still another technique is to use variable characters which suggest their role. This idea makes it much easier to follow a program, especially if it has been written by someone else.

After using these techniques to make a careful analysis of the errors in your program, the incorrect statement or statements may be corrected in any of these three ways:

1. Changing a line by retyping it correctly with the same line number.

2. Inserting a line by typing it with a new line number.

3. Deleting a line by typing only the line number.

We note that to be able to insert a line requires that the original line numbers NOT be consecutively numbered. For this reason, most users will number their line numbers in multiples of five or ten.

These corrections can be made at any time, that is, before or after you have run your program.

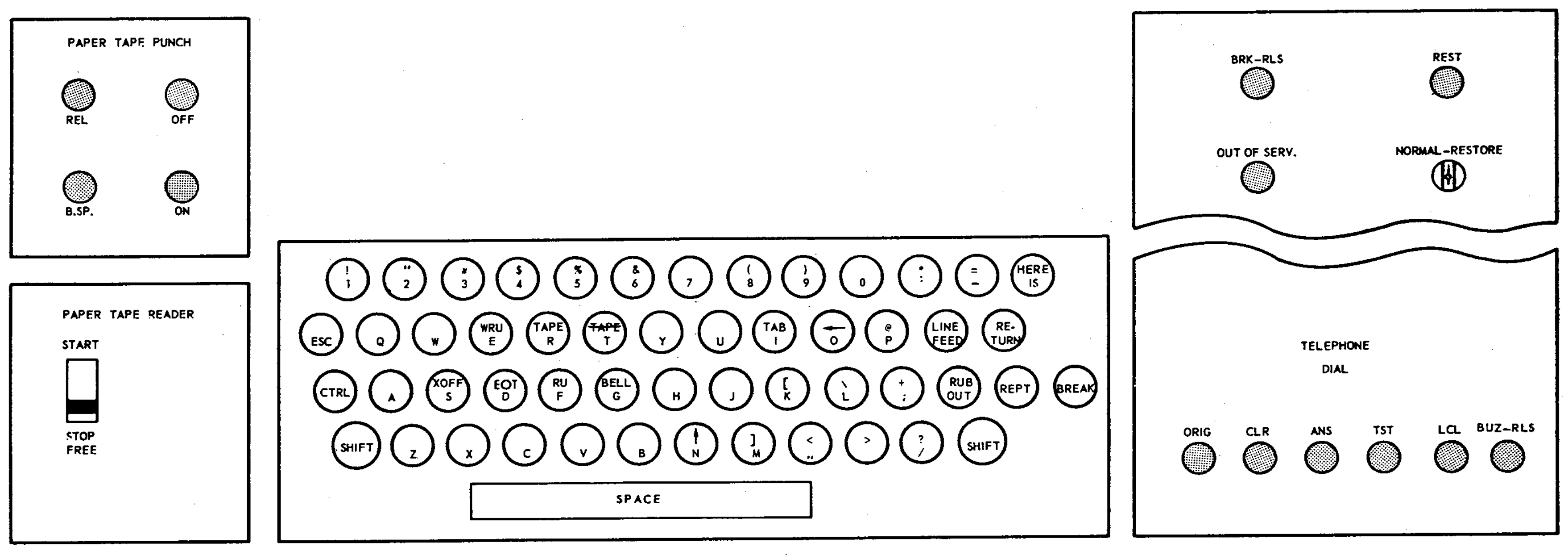
Since the computer sorts lines out and arranges them in order, a line may be retyped out of sequence. Simply retype the incorrect line using its original line number. The computer will replace the original incorrect line with this new correction. After many corrections have been made, you will find it most helpful to have the computer list out your current revised program for you to continue working with.

6. TELETYPE KEYBOARD AND SPECIAL KEYS

Special Keys on Keyboard

KEY	PURPOSE
Shift	Provides access to special characters and symbols.
Return	Carriage return - NOTE: The computer ignores the line being typed until this key is pushed.
← (Backward Arrow)	When pressed with the "shift" key, erases the last character typed or two presses erases the last two characters typed, etc.
Escape or ALT MODE	To erase an entire typed line before a "RETURN" is given.
Break	To terminate the running of your program.
Linefeed) Rubout) Rept)	Are used in off-line paper tape operations.

Model 33 Keyboard and Teletype Unit Controls



NOTE: ESC Key may be coded ALT MODE

Special Keys to Operate the Teletype

BUTTON	PURPOSE
ORIG	Turns on the Teletype machine.
CLR	Turns off the Teletype.
BUZ-RLS	Turns off buzzer, which goes on when the paper supply is low.
LCL (Local)	This mode cannot connect you with the computer. It is used for off-line tape punching.
BRK-RLS (Break Release)	Button is depressed to free keyboard after a "break" signal which locks the keyboard.

7. PROCEDURE FOR CONNECTING TO COMPUTER

For 265 Computer

1. Push "ORIG" button on right of teletype.

2. Dial number of computer. 265 # _________ 420 # _________.

3. After computer answers the sign on, sequence will automatically begin.
Type in your user number when the computer request. User Nr. _________.
PUSH "RETURN" after every line you type in.
Type in BASIC if the computer asks for the system you want.
NEW-OLD; OLD to access program previously stored.
    NEW to enter your new program.
Problem name must conform to the format given in the SOP (Appendix D) in this supplement.
    (Any 4 characters plus 2 of your dept code. First character must be alphabetic.)

4. "READY" is typed by the computer.

5. Entering program and/or information.

    a. To enter program and/or information directly with teletype:

    (1) Type in each Statement line beginning with a "line number" and ending with a "RETURN".

    (2) After the "END" Statement, which will have the largest line number, has been typed in and a "RETURN" given, type in "RUN" and give a "RETURN". This tells the computer to execute your program.

b. To enter information and/or program from paper tape:

(1) Type "TAPE".

(2) Have tape mounted in reader and after "READY" is printed, push lever up to START.

(3) When tape has reached final rubout area at end of tape, push lever down to manual cut off.

(4) Type "KEY" to return control to the keyboard.

(5) "READY" will be typed. Your program is in memory and is ready to be "RUN", "SAVE", "LIST", etc.

(6) To stop program at any time while COMPUTING, type "STOP". To stop program at any time while PRINTING, push "CONTROL" (CTRL), "SHIFT" and "P" keys all at the same time.

(7) To sign off -- type "BYE".

For 420 Computer

The Sign-on procedure is the same as 265 system with the addition of a pass-word control, which you will follow directly after the computer ask for your "user number".

Computer Types	You Type
i.e. PASSWD/CHG# MMMM XXX	Pass-word plus three characters

8. PAPER TAPE OPERATIONS

The user should plan his session at the teletype to avoid excess amounts of terminal connection time. One of the best methods to do this is to prepare as much of your work Off-Line on punch paper tape as possible. Remember the motto;

TYPING IS NO SUBSTITUTE FOR THINKING.

To Punch Paper Tape Off-Line:

1. Push "LCL" button on right of teletype.

2. Turn on the paper tape punch.

3. Hold both "RUBOUT" and "REPT" keys down until about 1Ø rubouts have been punched in the tape.

4. Type first line of program - starting with a line number.

5. At the end of every line punch "RETURN", "LINE FEED", AND "RUBOUT" in that order.

6. When finished - put about 2Ø rubouts at the end of the tape. Tear off tape through rubout area.

7. Turn off the paper tape punch and the teletype.

Punching Paper Tape from Computer

1. It is assumed that you are connected to the computer and that you are in "READY" status.

2. Turn on the paper tape punch.

3. Put about 1Ø rubouts on the paper tape.

4. Turn off the paper tape punch.

5. Type "LIST".

6. Timing is crucial now. After the title line has been printed and before the first line of the program is printed ---
turn on the paper tape punch.

7. The first line punched in the paper tape should be the first line of your program - starting with a line number.

8. When finished - put about 2Ø rubouts at the end of the tape and tear off tape through rubouts.

9. Turn off the paper tape punch.

APPENDIX A--LIMITATIONS ON BASIC

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

265 SYSTEM

Item	Limitation
Length of program	The total number of characters including carriage returns and spaces in the source BASIC program must not exceed 6248 characters.
Constants	The total number of different constants must not exceed 75.
Data	There can be no more than 128Ø data numbers.
FOR statements	There can be no more than 16 FOR statements in a program.
GOTO and IF-THEN statements	The total number of these statements combined cannot exceed 8Ø.
Lists and Tables	The total number of elements in all the lists and tables combined cannot exceed something less than 2ØØØ.

400 SYSTEM

Item	Limitation
Length of program	The maximum length of a source (BASIC) program is approximately 13Ø,ØØØ characters. Note that while a maximum size source program would not be expected to fit in the maximum size limitation for object programs, almost unlimited space is available for remark statements.
Data	There can be no more than 2Ø,ØØØ data numbers.

Lists and Tables	Similar to 265 system. The total number of elements in all lists and tables cannot exceed something less than 2ØØØ.
File (Program or data) Names	Special characters must not be used in the name assigned to a file. Several of the special characters (including the comma (,) and the slash (/) have been reserved as arguments to system commands.

APPENDIX B--SYSTEM COMMANDS

Alt Mode*	To delete an input line as if nothing had been typed.
← (Backward Arrow)	To erase the last character(x) typed. SHIFT key must also be depressed.
BASIC	To denote programming language.
Break*	To cause the computer to stop whatever it is doing with the program when printing is occurring.
BYE	To disconnect from the system.
CATALOG	To list a user's catalog of saved programs.
EDIT DELETE	To erase portions of a program.
EDIT EXTRACT	To retain portions of a program.
EDIT MERGE	To combine saved files into working store and to re-sequence line numbers.
EDIT RESEQUENCE	To resequence line numbers in program in working store.
EDIT WEAVE	To combine saved files into working store without re-sequencing.
GOODBYE	To disconnect from the System.
HELLO	To change user number or system.

*Control key on Teletype unit.

KEY	To reset terminal operation to normal after reading in paper tape.
LENGTH	To request the number of characters in working copy of program.
LIST	To list the current working copy of a program.
LIST:XX	To list the current working copy of a program beginning at line X or after (XXX 1 to 5 digits).
NEW	To introduce a new program and destroy the working copy of the current program.
OLD	To retrieve from saved store a previously saved file and destroy the working copy of the current program.
RENAME	To change program name but not working copy contents.
Return*	To terminate a program line, cause the System to take action based upon input provided, and act as a normal carriage return.
RUN	To compile and execute. Also to determine elapsed run time.
SAVE	To save permanently the working copy of a program.
SCRATCH	To eliminate from the working copy of a program everything but the program name.
STOP	To cause the computer to stop whatever it is doing with the program except when printing is occurring.
TAPE	To inform the System that paper tape will be read in.
UNSAVE	To release and destroy a previously saved program.
User Number	Four to six characters that identify the user to the system.

*Control key on Teletype unit.

APPENDIX C--265 ERROR MESSAGES--265 SYSTEM

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

Error Message	Interpretation
DIMENSION TOO LARGE	The size of a list or table is too large for the available storage. Make them smaller. (See Appendix A)
ILLEGAL CONSTANT	More than nine digits or incorrect form in a constant number, or a number out of bounds ($>$5.78960 E 76).
ILLEGAL FORMULA	Perhaps the most common error message, may indicate missing parentheses, illegal variable names, missing multiply signs, illegal numbers, or many other errors. Check the statement thoroughly.
ILLEGAL RELATION	Something is wrong with the relational expression in an IF-THEN statement. Check to see if you used one of the six permissible relational symbols.
ILLEGAL LINE NUMBER	Line number is of incorrect form, or contains more than five digits.
ILLEGAL INSTRUCTION	Other than one of the fifteen legal BASIC instructions has been used following the line number.
ILLEGAL VARIABLE	An illegal variable name has been used.
INCORRECT FORMAT	The format of an instruction is wrong. See especially IF-THEN's and FOR's.
END IS NOT LAST	Self-explanatory, it also occurs if there are two or more END statements in the program.

NO END INSTRUCTION	The program has no END statement.
NO DATA	There is at least one READ statement in the program, but no DATA statements.
UNDEFINED FUNCTION	A function such as FNF ( ) has been used without appearing in a DEF statement. Check for typographical errors.
UNDEFINED NUMBER	The statement number appearing in a GOTO or IF-THEN statement does not appear as a line number in the program.
PROGRAM TOO LONG	Either the program itself is too long for the available storage, or there are too many constants. (See Appendix A.)
TOO MUCH DATA	There is too much data in the program. (See Appendix A.)
TOO MANY LOOPS	There are too many FOR-NEXT combinations in the program. The upper limit is 26. (See Appendix A.)
NOT MATCH WITH FOR	An incorrect NEXT statement, perhaps with a wrong variable given. Also, check for incorrectly nested FOR statements.
FOR WITHOUT NEXT	A missing NEXT statement. This message can also occur in conjunction with the previous one.
CUT PROGRAM OR DIMS	Either the program is too long, or the amount of space reserved by the DIM statements is too much, or a combination of these. This message can be eliminated by either cutting the length of the program, or by reducing the size of the lists and tables, reducing the length of printed labels, or reducing the number of simple variables.

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

OUT OF DATA	A READ statement for which there is no DATA has been encountered. This may mean a normal end of your program, and should be ignored in those cases. Otherwise, it means that you haven't supplied enough DATA. In either case, the program stops.
SUBSCRIPT ERROR	A subscript has been called for that which lies outside the range specified in the DIM statement, or if no DIM statement applies, outside the range Ø through 1Ø. The program stops.
INPUT DATA NOT IN CORRECT FORMAT	Input from Teletype consists of an illegal number format, wrong format, or wrong number of variables.
RETURN BEFORE GOSUB	Occurs if a RETURN is encountered before the first GOSUB during the running of a program. (Note: BASIC does not require the GOSUB to have an earlier statement number--only to perform a GOSUB before performing a RETURN.) The program stops.
DIVISION BY ZERO	A division by zero has been attempted. The computer assumes the answer is +∞(about 5.7896ØE76) and continues running the program.
ZERO TO A NEGATIVE POWER	A computation of the form Ø↑(-1) has been attempted. The computer supplies +∞ (about 5.7896ØE76) and continues running the program.
ABSOLUTE VALUE RAISED TO POWER	A computation of the form (-3)↑2.7 has been attempted. The computer supplies (ABS(-3)) 2.7 and continues. Note: (-3)↑3 is correctly computed to give -27.
OVERFLOW	A number larger than about 5.7896ØE76 has been generated. The computer supplies + (or -) ∞ (about ± 5.7896ØE76) and continues running the program.

UNDERFLOW	A number in absolute size smaller than about 4.318Ø9E-78 has been generated. The computer supplies Ø and continues running the program. In many circumstances, underflow is permissible and may be ignored.
EXP TOO LARGE	The argument of the exponential function is 176.753. + ∞ (5.7896ØE76) is supplied for the value of the exponential and the running is continued.
LOG OF NEGATIVE NUMBER	The program has attempted to calculate the logarithm of a negative number. The computer supplies the logarithm of the absolute value and continues.
LOG OF ZERO	The program has attempted to calculate the logarithm of Ø. The computer supplies - ∞ (about -5.7896ØE76) and continues running the program.
SQUARE ROOT OF A NEGATIVE NUMBER	The program has attempted to extract the square root of a negative number. The computer supplies the square root of the absolute value and continues running the program.

APPENDIX C--400--ERROR MESSAGES--400 SYSTEM

Some of these messages occur during compilation and others during execution of a program. Many of the messages not only identify the type of error, but indicate the line number where the error occurred. It is expected that as the development of the BASIC language continues these error messages will be revised.

During execution, certain messages occur which do not stop execution, but inform the user of irregular conditions existing in identified lines of his program. Other messages, however, point out serious errors which stop execution.

COMPILATION ERRORS

BAD FUNCTION CALL LINE # xxxxx	A function such as FNF (X) has been used without appearing in a DEF statement. Check for typographical errors.
CUT # OF LINES OR VARIABLES	Too many lines or variables have been used. Reduce the number of statements or variables.
CUT PROGRAM OR ARRAY SIZE	Either the program is too long, the amount of space reserved by the DIM statements is too much, or a combination of these. This message can be eliminated by cutting the length of the program, reducing the size of list and tables, or reducing the length of printed labels.
DIM TOO LARGE LINE # xxxxx	The size of an array exceeds the available storage.
DIMENSION ERROR LINE # xxxxx	A dimension error or inconsistency has occurred in connection with an array or matrix.
END IS NOT LAST	Self-explanatory; it also occurs if there are two or more END statements in the program.
EXPRESSION TOO COMPLICATED LINE # xxxxx	Too many operations have been attempted in a single expression. Use two or more simpler expressions instead.
FILE ERROR LINE # xxxxx	This covers a variety of errors, usually in the form of the file name, user number, or library name.

FILE IN USE LINE # xxxxx	The file which was referenced is already being used.
FILE NOT SAVED LINE # xxxxx	The file was not saved under the requested user number.
FILE NOT SAVED FOR READ OR WRITE LINE # xxxxx	The file was not saved with the proper privileges (READ, WRITE, APPEND). This will occur in the first reference to the file even though the file may be saved with the proper privileges for that operation.
FILE OUT OF DATA LINE # xxxxx	The input file ran out of data; line number specified is that of the program INPUT statement.
FILE OPERATION ERROR LINE # xxxxx	A READ and WRITE was attempted without restoring or closing in between.
FOR NOT MATCHED WITH NEXT LINE # xxxxx	A NEXT statement is missing.
FOR-NEXT LOOPS NESTED TOO DEEPLY LINE # xxxxx	There are too many FOR-NEXT combinations in the program. The upper limit is 26.
ILLEGAL ARRAY NAME LINE # xxxxx	An illegal array name has been used.
ILLEGAL CONSTANT LINE # xxxxx	An illegal constant has been used. It is more than nine digits, in an incorrect form, or a number out of bounds ($>$5.7896E76).
ILLEGAL EXPRESSION LINE # xxxxx	This may indicate missing parentheses, illegal variable names, missing arithmetic symbols, illegal numbers, or many other errors. Check the statement thoroughly.
ILLEGAL FUNCTION CALL LINE # xxxxx	This may indicate missing parentheses, misspelled function name, missing equal sign, or many other errors.
ILLEGAL LINE NUMBER LINE # xxxxx	Referenced line number is of incorrect form, or contains more than five digits.
INPUT FILE ERROR LINE # xxxxx	Contents of the input file are something other than line numbers, digits, literals or commas.

LEFT PAREN NOT MATCHED WITH RIGHT LINE # xxxxx	A left parenthesis has been processed without a corresponding right parenthesis. Check the parentheses count thoroughly.
NEXT NOT MATCHED WITH FOR LINE # xxxxx	An incorrect NEXT statement, perhaps with a wrong variable given. Also, check for incorrectly nested FOR statements or an omitted FOR statement.
NO DATA	There is at least one READ statement in the program, but no DATA statements.
NO END STATEMENT	The program has no END statement.
NO PROGRAM	There is no program.
OPEN QUOTE LINE # xxxxx	The quote that terminates a message is missing.
PROGRAM BEING USED; TRY LATER	A program exercises exclusive control over a file since the program may update or otherwise alter the file. This message means that some other user has control of the file you are requesting and you must wait until his program releases control.
STRING VARIABLE TOO LONG LINE # xxxxx	The assignment of a string variable with more than 15 characters has been attempted.
TOO MANY FILES OPEN LINE # xxxxx	The limit of 5 open files at the same time has been exceeded.
TOO MUCH DATA AT LINE # xxxxx	The total amount of data is too much for the available storage. This message can be eliminated by reducing the amount of data.
UNDEFINED ARRAY NAME LINE # xxxxx	An array name has been used, but was not defined.
UNDEFINED FUNCTION CALL LINE # xxxxx	A function such as FNF(X) has been used without appearing in a DEF statement.
UNDEFINED LINE NUMBER LINE # xxxxx	The referenced line number appearing in a GOTO, ON, or IF-THEN statement does not appear as a line number in the program.
UNRECOGNIZABLE STATEMENT LINE # xxxxx	Other than one of the legal BASIC instructions has been used following the line number.

WRONG PASSWORD LINE # xxxxx	The password was incorrect. This can occur if the file is saved without a password; and when the file is referenced, a password is used.

EXECUTION ERRORS

CUT STRING VARIABLES	Either the amount of space used by string variables exceeds the available storage, or the program is too long. Reduce the size of the string variables or the size of the program.
DIVISION BY ZERO	A division by zero has been attempted. The computer assumes the answer is $+\infty$ (about 5.7896ØE76) and continues running the program.
EXP LINE # xxxxx(x)	The argument of the exponential function $\geqslant$176.753. $+\infty$ (About 5.7896ØE76) is supplied for the value of the exponential, and the running is continued.
FILE OUT OF DATA LINE # xxxxx	An INPUT statement has been encountered for a file which has no data. This may mean a normal end of the program, and in that case, should be ignored. Otherwise, it means that insufficient data was supplied. In either case, the program stops.
GOSUB NESTED TOO DEEPLY LINE # xxxxx	Too many GOSUBs without a RETURN. It may mean that subroutines are exiting by GOTO or IF-THEN statements rather than by RETURNs. The program stops.
INCORRECT FORMAT-RETYPE	The input data is in an incorrect format. The data must be retyped.
INPUT FILE FORMAT ERROR LINE # xxxxx	The input file contains characters other than digits and commas. Correct the data on the input file.
LOG LINE # xxxxx(x)	The program has attempted to calculate the logarithm of a negative number. The computer supplies the logarithm of the absolute value and continues; or the program has attempted to calculate the logarithm of Ø. The computer supplies $-\infty$ (about -5.7896ØE76) and continues running the program.

NEARLY SINGULAR MATRIX LINE # xxxxx	The INV operation in a MAT statement has encountered a matrix with zero or nearly zero pivotal elements. The matrix being inverted is singular or nearly so.
ON LINE # xxxxx	The value of the expression in an ON statement, when rounded, must be an integer which is greater than zero but less than or equal to the number of line numbers following the expression.
OUT OF DATA LINE # xxxxx	A READ statement for which there is no DATA has been encountered. This may mean a normal end of the program, and in that case should be ignored. Otherwise, it means that the user has not supplied enough data. In either case, the program stops.
OVERFLOW LINE # xxxxx	A number larger than about 5.7896ØE76 has been generated. The computer supplies + (or -) ∞ (about + or -5.7896ØE76) and continues running the program.
RETURN BEFORE GOSUB LINE # xxxxx	Occurs if a RETURN is encountered before the first GOSUB during the running of a program. (note: BASIC does not require the GOSUB to have an earlier statement number--only to perform a GOSUB before performing a RETURN.) The program stops.
SIN ARG LINE # xxxxx	The argument of the sine or cosine function exceeds the limit of 1.374E11. Zero is supplied for the value of the function, and running of the program continues.
SQR LINE # xxxxx(x)	The program has attempted to extract the square root of a negative number. The computer supplies the square root of the absolute value and continues running the program.
STRING VARIABLE TOO LONG LINE # xxxxx	The string variable has exceeded the limit of 15 characters. The program stops.
SUBSCRIPT ERROR LINE # xxxxx A(x,y)	A subscript has been called for that which lies outside the range specified in the DIM statement, or if no DIM statement applies, outside the range Ø through 1Ø. The program stops.

TAN LINE # xxxxx	The argument of the tangent function exceeds the limit of 1.374E11. + ∞(5.7896ØE76) is supplied for the value of the function, and running of the program continues.
UNDERFLOW LINE # xxxxx	A number in absolute size smaller than about 4.318Ø9E-78 has been generated. The computer supplies Ø and continues running the program. In many circumstances, underflow is permissible and may be ignored.

APPENDIX D--STANDING OPERATING PROCEDURES

REMOTE COMPUTER TIME-SHARING TERMINAL

TABLE OF CONTENTS

SECTION I - GENERAL

PART A - PURPOSE AND OBJECTIVES

1. Purpose of Standing Operating Procedures

The purpose of this SOP is to promulgate the policies necessary to insure efficient use of the computer-time-sharing terminal at the US Army Engineer School. The monitoring agency will be the Instructional Data Systems Project Office (IDSPO).

2. Objectives of the Computer Time-Sharing System

The remote terminal at the Engineer School will serve two functions. Primarily, it will assist in the development of computer applications to be used in the generation of systems specifications necessary for the procurement of our own computer at the Engineer School. Secondly, it will provide faculty familiarization with and training in the use of the CAI educational mode.

PART B - SYSTEM DESCRIPTION

Definition of Time-Sharing

Computer time-sharing is, as the name implies, the sharing of a computer by many users. With modern third generation computer hardware and related software, it is possible to execute simultaneously several computer programs and to run an even larger quantity of programs concurrently. Thus, with a large computer system, it is feasible to allow almost unlimited access by an extremely large number of users. The Engineer School has procured and is in the process of procuring, contracts with several firms that specialize in computer time-sharing services. Through the teletype terminal located in Humphreys Hall, the user will have access to all these facilities.

PART C - AUTHORIZED PERSONNEL

1. Faculty

Any faculty member can have almost unlimited access to the teletype terminal. However, before a faculty member will be allowed this freedom, he must take a brief (2 hour) orientation course conducted by the Instructional Data Systems Project Office. Faculty members can develop program applications on the terminal at their own discretion; but before actually attempting to use a CAI application in the classroom, the instructor must first consult the Educational Advisor to make a determination as to the utility of the application.

2. Students

Initially, students will not be allowed access to the terminal. Experimental projects may be conducted allowing students to use the terminal, but these will be quite limited and closely controlled by the Instructional Data Systems Project Office.

SECTION II - USAGE

PART A - SIGN IN PROCEDURE

1. <u>Use of Log</u>

a. The Remote Terminal User's Log (USAES Form 73) is intended to provide the IDSPO with pertinent statistical data in order to correlate the monthly charges and to determine the usage of the terminal.

b. Each time the remote terminal is used and contact is made with the computer, the user will make the appropriate entries on the log. The user will enter his name, rank, and social security account number. Then he will enter the title of his program with a brief discription of the program. The program name will conform to the following format:

XXXX10	D/E&MS
XXXX30	D/M&TE
XXXX50	D/TOPO
XXXX70	IDSPO

where X is any alphabetic character. The user will then enter the time and date prior to running his program. After he is finished at the terminal, he will enter the total terminal connect time used and the total running (CPU) time used. The remarks column will contain the name of the computer service used (CEIR, COMNET, etc), how the terminal was used (de-bugging of program, problem solving, classroom instruction, etc.), and state whether the program was saved or not. If the service used has more than one system, include which system was used; e.g., CEIR (265) or CEIR (420). An example of the Remote Terminal User's Log is shown on the next page. All entries will be printed and legible!

PART B - STORAGE OF PROGRAMS AND PROGRAM FORMAT

1. <u>When to Store Programs</u>

The Engineer School will be assessed storage fees for programs stored in any of the available systems. Since the School is restricted by a limited budget for these computer services, it is mandatory that the number of stored programs be kept to a minimum. Any program may be temporarily stored; otherwise, all programs will be kept on paper tape. When a program is stored, it will be noted on the log. If no such notation is made, the program will be purged from the system's storage library.

2. <u>Paper Tapes</u>

Punched paper tapes should be used to the fullest extent possible to maintain program files. The Instructional Data Systems Project Office will keep a documented library of all tapes which will be of general engineering interest. Copies of any programs which will be of such a nature, should be given to IDSPO to be included in the library.

3. <u>Program Library</u>

When a program has been de-bugged, a copy of it along with a sample run should be sent to IDSPO. These programs are necessary to determine workload requirements in order to write system specifications. They will also be made available to other users.

4. <u>Program Format</u>

In order to facilitate the processing of Automatic Data Processing applications, several comment statements will be added to the beginning of each program. These comments will include the programmer's name, office, phone number, and a narrative of the program to include a description of the variables used. Where the system allows, a "LENGTH" command, or its equivalent, should be given and included with the sample run given to the IDSPO.

PART C - RESERVING THE TERMINAL

<u>How and Where</u>

The terminal will have receptacles in Room 118 and in classrooms 1A, 1B, 1E, and 1F of Humphreys Hall. It can be used in any of these rooms when they are not being used. The terminal itself can be reserved by signing the appointment sheet attached to the teletype unit. When available, one of the classrooms can also be reserved up to 3 weeks in advance by contacting Schedules Branch, Operations Division, Director of Instruction (42973). If the terminal and a classroom are needed on a recurring basis (i.e., for classroom instruction, etc.) contact the Operations Officer, IDSPO (42897).

PART D - ECONOMY IN THE USE OF THE SYSTEM

<u>The Purpose For and the Means of Economy</u>

a. The Engineer School has been allocated a limited amount of funds for ADP services. Much has to be accomplished with these resources. Consequently, emphasis will be placed on the efficient use of the remote time-sharing terminal. Programs should be punched on paper tape off-line and then read intothe system on-line. Entire programs will not be typed on-line. Programs will be stored only as outlined previously.

b. The School is charged for the time while the terminal is connected to the computer; consequently, sessions on the teletype terminal will be carefully planned. CAI applications may be tested in the classroom, but lack of funds will prevent their use on a permanent basis.

NOTE: Any changes to this SOP will be included with the copy that is attached to the terminal and notice of such changes will be given. When a notice of a change has been posted, the user must read these revisions before continuing his use of the terminal.

APPENDIX E--ADVANCE BASIC TECHNIQUES

Advance BASIC Techniques will be available at a later date.