SCIENTIFIC DATA SYSTEMS

Reference Manual

SDS 940 CAL

Price: $1.50

CAL
REFERENCE MANUAL

for

SDS 940 TIME-SHARING COMPUTER SYSTEMS

June 1967

90 11 14A

SDS

SCIENTIFIC DATA SYSTEMS/1649 Seventeenth Street/Santa Monica, California

Printed in U.S.A.

RELATED PUBLICATIONS

Title	Publication Number
SDS 940 Computer Reference Manual	90 06 40
SDS 940 Time-Sharing System Reference Manual	90 11 16
SDS 940 Terminal User's Guide	90 11 18

CONTENTS

1. INTRODUCTION 1

Operating Procedures 1
Log In 1
Escape 1
Exit and Continue 2
Log Out 2

2. BASIC CONCEPTS 2

Number Representation 2
Variables 3
Arrays and Array Elements 3
Vectors and Vector Elements 3
The Concept of a Two- or Three-Dimensional Array 4
Expression 5
Arithmetic Operations 5
Interpreting the Order of Computation (Precedence) 6
Parentheses in CAL Expressions 6
Executing Single Commands 6
The "TYPE" Command 6
Errors 8
Error Diagnostics 8
The "SET" Command 8
Mathematical Functions 9

3. CAL PROGRAMS 11

Steps and Parts 11
Adding or Deleting Steps 11
Control Characters 12
The "EDIT" Command 14
"FOR" and "DO" 14
The "FOR" Clause 14
The "DO" Command 15
Example of a Hypotenuse Program Using the "DO Step" Command 16
Decision Commands and Comparison Operators 16
The "IF" Clause 16
Comparison Operators 17
Boolean (Logical) Operators 18
Precedence in the Boolean Operators 18
Negative Numbers with Boolean Operators 18
The "TO" Command 20
Modifying Clauses Other than "IF" and "FOR" 20
Use of the "UNLESS" Clause 20
Use of the "WHILE" Clause 20
Use of the "UNTIL" Clause 21
Multiple Clauses 21
Conditionals as Expressions 21
The "WHERE" Modifier 22

4. SPECIAL STATEMENTS (INCLUDING FILE CONSTRUCTION) AND SUBPROGRAMS 23

Special Statements 23
Special Direct Statements 23
Special Indirect Statements 25
Subprograms 26
One-Step Subprograms 26
Multiple-Step Subprograms 27
Program Files 28

5. INPUT/OUTPUT – DATA FILE CONSTRUCTION 29

The "DEMAND" Statement 29
"DEMAND" Statements with Modifiers 29
The "FORM" Statement for Input 29
Data Input from Paper Tape 30
The "FORM" Statement for Output 30
Input from Files 32

APPENDIX

CAL SUMMARY 33

Numbers 33
Variables 33
Arithmetic Operators 33
Logical Operators 33
Mathematical Functions 33
Commands (Direct or Indirect Statements) 33
Commands (Direct Statements Only) 33
Modifiers 34
Edit Characters 34

FIGURES

1. Examples of CAL Treatment of Numbers 2
2. Legal and Illegal CAL Variables 3
3. Column and Row Vector Notation in CAL 4
4. Hypothetical Major City Food-Price Matrix 4
5. $\overline{P}$ Array and CAL Notation in Two Dimensions 5
6. Illustrations of the Mathematical Operator Precedence Sequence 7
7. CAL Expression Using Parentheses and Mathematical Operator Precedence 7
8. Example of a CAL Program 11
9. Hypotenuse Problem 12
10. Truth Table for Steel Grading 19
11. Sample Assembly Parts Bill of Material List 21
12. Value of A as a Function of X 22

PREFACE

This manual is intended to serve as a tutorial guide for the new CAL user and as a reference source for the experienced user. For clarity, several typographic conventions are used throughout the manual.

1. Underscored copy in an example represents that produced by the system in control of the computer. Unless otherwise indicated nonunderscored copy in an example is that typed by the user.

2. The notation (RET) signifies a carriage return and (LF) signifies a line feed. The user signals the end of a CAL command by striking the RETURN key, and the system confirms the command with an automatic line feed. If either or both of these functions are initiated by the system, no notation is used (in most of the examples given).

1. INTRODUCTION

The SDS 940 Conversational Algebraic Language (CAL) permits the user to solve mathematical problems while connected to the computer via Teletype.

Typical problems that can be solved with CAL include numeric ones that do not require a great deal of computation and many kinds of problems involving non-numeric processing. In many cases, CAL will identify errors made by the user and will provide examples to illustrate the required format.

A CAL program consists of a series of statements referred to as "steps". Each step is a component part of the problem solution. All the steps combined constitute the programmed problem solution. Once a program has been written, it can be saved and re-executed indefinitely.

The salient capabilities of the CAL language are:

1. Basic arithmetic operations (i.e., addition, subtraction, multiplication, division, and exponentiation).
2. Relational operators (i.e., =, #, >, and <).
3. Logical operators (i.e., AND, NOT, and OR).
4. Conditional expressions (e.g., IF, THEN, WHILE, UNTIL, and UNLESS).
5. Mathematical function subroutines (e.g., trigonometric functions, logarithms, and square root).
6. Editing by character, word, or line.
7. Data file manipulation instructions allowing the user to create, save, read, write, and delete data files.
8. Simple I/O instructions.
9. Language diagnostics.

OPERATING PROCEDURES

The standard procedure for gaining access to an SDS 940 time-sharing computer center from a remote Teletype terminal is described in the SDS 940 Terminal User's Guide, Publication No. 90 11 18. The following paragraphs summarize the standard procedures as they apply to CAL users.

LOG IN

To gain access to the computer, the following operating sequence is performed:

1. If the FD-HD (Full Duplex-Half Duplex) switch is present, turn the switch to FD. This is a toggle switch with two locking positions. When the Teletype is not connected to the computer (sometimes called the Local Mode), this switch must be in the HD position. Whenever the Teletype is connected to the computer, this switch must be in the FD position.

2. Press the ORIG (originate) key, which is usually located at the lower right corner of the console directly under the telephone dial, to obtain a dial tone before dialing the computer center.

3. Dial the computer center number. When the computer accepts your call, the ringing will change to a high-pitched tone. Then, a request that the user log in will appear on the Teletype.

   PLEASE LOG IN!

4. The user must then type his account number, password, and name in the following format:

   number password;name

   Only persons who know all three elements (the account number, password, and name) may log in under that particular combination. The following examples all illustrate acceptable practice:

PLEASE LOG IN!	A7PASS;JONES
PLEASE LOG IN!	C8WORD;BROWN
PLEASE LOG IN!	F5PW;PSEUDO

   If the account number, password, and name are not recognized by the computer, it will print INVALID USER. The password is not printed. The log-in procedure must then be repeated. If the user does not type his account number, password, and name within a minute and a half, a message is transmitted instructing him to call the computer center for assistance. The computer will then disconnect the user and the dial and log-in procedure will have to be repeated.

5. If the account number, password, and name are accepted by the computer, it will print READY, the date, and the time on the next line. The user must then depress the carriage return key, after which the computer responds by printing a dash at the beginning of the following line:

   READY date, time

   The dash indicates that the 940 Executive is ready to accept a command.[†]

   In response to the dash, the user types

   CAL

   When ready to accept commands, CAL responds with a carriage return and line feed, and then prints CAL.

ESCAPE

The ESCAPE (ESC) key[††] may be used at almost any time. It causes the subsystem to abort the current operation and ask for a new command. Striking the (ESC) key before terminating a command with (RET) aborts the command.

[†] In some 940 time-sharing systems the commercial "at" sign, @, is used to indicate that the 940 Executive is ready to accept a command.

[††] In some 940 time-sharing system configurations the RUBOUT or ALT MODE key is used instead of the ESCAPE key. Where (ESC) appears in this manual, RUBOUT or ALT MODE may be substituted.

EXIT AND CONTINUE

Striking the ESC key twice in succession causes computer control to return to the Executive. The user may reenter CAL with program and execution status intact by typing

-CONTINUE RET

The system then prints a "CAL", to confirm the action, and returns control to CAL.

LOG OUT

When the user wants to disconnect, he types two consecutive escapes (to return control to the Executive) and then types

LOGOUT

The computer will respond by printing the amount of computer and hook-up (line) time charged to the user's account since the previous log-in procedure was completed.

2. BASIC CONCEPTS

NUMBER REPRESENTATION

There are three ways to represent numbers in CAL. First, numbers may be expressed as integers; i.e., whole numbers. This means that they have no fractional part, e.g., 1, 5, 22. Next, numbers may be expressed in floating point form, e.g., 1.5, 3.0, 99.9. The third way of expressing a number is by scientific notation. For example, the number 62,000,000,000,000,000,000,000,000,000,000 is equal to 62E30 in scientific notation. The "E" in this notation means that the number to the left of the E (which may be in integer or floating point form) is multiplied by ten raised to the power specified by the number to the right of the E (which may be either a positive or a negative number).

CAL will accept as input all three types of numbers, but with limitations of size. Integers will be accepted as long as they have less than eight digits[†]. If the integer has more than seven digits, it must be expressed in scientific notation. Floating point numbers may be input if they have no more than seven digits total. A floating point number with more than seven digits must be expressed in scientific notation.

In all the cases, numbers in CAL may be thought of as being in floating point form. (An integer is always assumed to have a decimal point, even if it is not specified, e.g., 5 is assumed to be 5.0). This makes it possible for the computer to perform arithmetic operations (add, subtract, multiply, and divide) and position the decimal point automatically. Thus it is unnecessary for the user to scale and rescale as a problem progresses. The CAL compiler does not need to know what type of number the user is thinking about.

After calculation, CAL will print out numbers in one of the three forms. Examples are given in Figure 1. CAL prints out numbers as integers if they have no fractional parts and are smaller than 9,999,999. If an integer has more than seven digits, it is printed out in scientific notation. CAL prints out numbers with fractional parts in floating point form. If the floating point number has more than 7 significant digits, it will be rounded off.

Operator Input		CAL Output
TYPE	.0123	1.2300000-02
TYPE	26.3421	26.342100
TYPE	987654321	9.8765432 08
TYPE	300000000	3.0000000 08
TYPE	3000000	3000000
TYPE	3000000.	3000000
TYPE	30E6	30000000
TYPE	362E-2	3.6200000
TYPE	362E-10	3.6200000-08
TYPE	1234.00	1234
TYPE	.0023E-6	2.3000000-09
TYPE	.1234567E-20	1.2345670-21
TYPE	246↑12	4.9115807 28

Figure 1. Examples of CAL Treatment of Numbers

Note in the examples given in Figure 1 that CAL does not print an E in the numbers expressed in scientific notation. If a number has a blank preceding the last two digits, the number formed by the last two digits is understood to be a positive power. Negative exponents are preceded by a minus sign. The student is encouraged to try other combinations and sizes of numbers after reading this chapter.

[†]The eighth position is always reserved for the decimal point, present or not.

The computer automatically determines the form in which the number is printed out, unless otherwise directed by the user. The commands which make it possible for the user to designate the kind of output he desires are discussed in Chapter 5.

VARIABLES

A variable in CAL is a symbol for a number or value. Associated with the variable is a place in the computer memory where the number or value is stored. There are 260 distinct, named variables in CAL, one for each letter of the alphabet, optionally followed by a single digit (0-9). Since some problems may require more than 260 variables, a variable may be subscripted. The subscript is enclosed in parentheses. In most cases, subscripting will extend the number of available variables to considerably more than are ordinarily required to write a program. Examples of legal and illegal variables and subscripts are shown in Figure 2.

Legal Variables	B, C, A, A(1), A(N), A(N+1), A(N+M), A(1, 2), A(N, M), A(N+1, 2), A(N+1, M+1), X(23, 65, -147.3), X(A(1, 3)), A1, B2, X(A(N)+13, A(M+1))
Illegal Variables	AB, AN, 1B, 2B

Figure 2. Legal and Illegal CAL Variables

A subscript may be any arbitrary expression. It is truncated to the nearest, smallest integer before use (e.g., 4.65 would be taken as 4.0). A variable may have any number of subscripts, and it need not have the same number each time it is referenced. For example, "A" refers to a different number than A(0) or A(1). Consider the following example of a program to compute the amount to invoice for a number of items. The formula is assumed to be

(Quantity) · (Cost) · 1.20 = Invoice Amount

In CAL, the variables could be designated as

Quantity = Q

Cost = C

Invoice Amount = I

The CAL Statement would be

```
SET I = Q*C*1.20          (* means multiply in CAL)
```

Or, the variables could be designated as

A(1) = Quantity

A(2) = Cost

A = Invoice Amount

The CAL Statement would then be

```
SET A = A(1)*A(2)*1.20
```

Subscripted variables in CAL may also be used to construct arrays and to denote specific array elements rather then uniquely different and unrelated items in a program.

ARRAYS AND ARRAY ELEMENTS

In constructing a program, it is frequently convenient to deal with a group of variables which form or belong to a single class or collection. When the variables form an ordered set, they can be related to one another by subscript notation. Such a collection is referred to as an array and the variables belonging to this array as array elements. Sometimes the synonym matrix is used for arrays, and the elements are then called matrix elements.

For example, the set of numbers shown below would be a two dimensional array.

$$\begin{vmatrix} 1 & 8 & 1 \\ 2 & 4 & 2 \\ 3 & 2 & 3 \end{vmatrix}$$

This array could also be termed a 3-by-3 matrix.

VECTORS AND VECTOR ELEMENTS

A string of numbers that can be arranged in a single row or column is thought of as a one dimensional array. Such an array may be called a vector.[†] In most textbooks an element of a vector is generally written as a lower case letter with a subscript, e.g., a_j, b_k, x_j. The subscript denotes which element in the vector is indicated, the sequence being from left to right if written in a row, or from top to bottom if written in a column. To denote the entire vector, a lower case letter with a bar over the top is generally used, e.g., $\bar{a}$, $\bar{b}$, or $\bar{x}$. Row and column vectors in CAL are shown in Figure 3.

Each element in the vector is in itself a variable. Note that in expanding the variable name base by the use of subscripts, the user should be careful not to overlap an array using the same letter of the alphabet.

[†]As used here, vector does not necessarily imply a directed line segment such as that encountered in elementary mechanics. The term is borrowed from elementary mechanics and broadened to mean simply an ordered set of quantities.

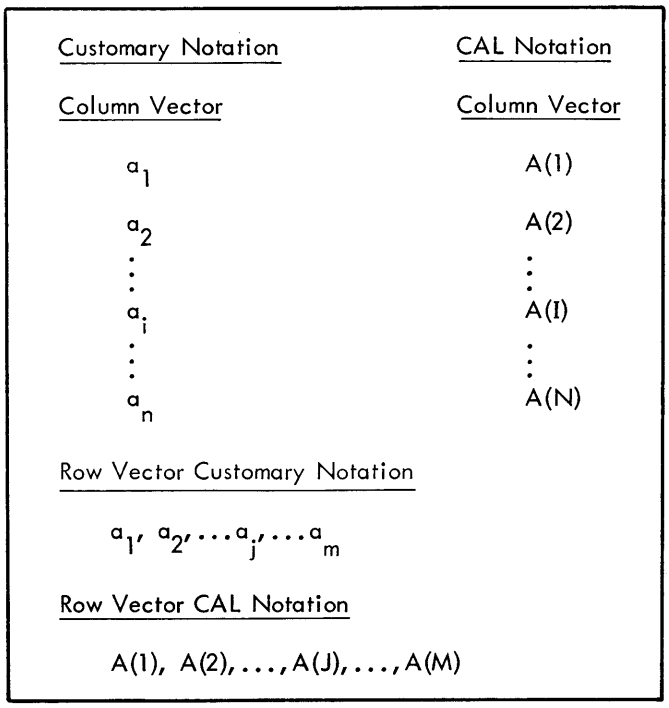


Figure 3. Column and Row Vector Notation in CAL

As an example of an array, consider the price of steak in several cities.

City	Implied Subscript	Steak Price per Pound
New York	1	$1.29
Detroit	2	$1.32
Pittsburgh	3	$1.18
Denver	4	$1.05
San Francisco	5	$1.35

If the vector were called $\overline{p}$, the variables containing the price of steak in New York, Detroit, Pittsburgh, Denver, and San Francisco would be P(1), P(2), P(3), P(4), and P(5) respectively, in CAL notation. The same variable base name, P, may be used for more than one related or unrelated item. Thus the price of butter in the same five cities could be designated by P(6) through P(10). If there were an overlap and the price of butter were to be entered in P(4), the price of steak in Denver, would be lost. To avoid confusion and such possible losses, the integrity of each individual variable must be maintained by the programmer in his bookkeeping of the subscript numbers.

THE CONCEPT OF A TWO— OR THREE— DIMENSIONAL ARRAY

If two subscripts are used to identify the elements of an array, the collection is referred to as a two-dimensional array or matrix. The pattern of squares on a checkerboard is an example of a two-dimensional array. In its most common usage a matrix is a rectangular array of quantities in which the rows (or columns) are related vectors. Alternately, we may say that a matrix is an ordered set of related vectors. A matrix is usually denoted by a capital letter with a bar over it, e.g., $\overline{A}$, $\overline{B}$, $\overline{C}$. An element of a matrix is usually represented by a lower case letter with a double subscript. Therefore, elements of the matrix $\overline{A}$ are represented by $a_{1,1}$, $a_{1,2}$, $a_{2,1}$, $a_{i,j}$, etc. The first subscript denotes the row that contains the element, while the second subscript denotes the column.

Using the previous example of the price of steak in several cities, we can add vectors indicating the price of bread, butter, and potatoes. A matrix made up of an ordered set of column vectors would then be as shown in Figure 4.

City	Steak	Butter	Bread	Potatoes
New York	$1.29	.84	.42	.08
Detroit	1.32	.86	.40	.07
Pittsburgh	1.18	.82	.39	.06
Denver	1.05	.79	.35	.05
San Francisco	1.35	.90	.44	.10

Figure 4. Hypothetical Major City Food-Price Matrix

If this matrix is called $\overline{P}$, the price of steak in Denver would be the element $p_{4,1}$; the price of potatoes in Pittsburgh would be the element $p_{3,4}$; etc. (For corresponding CAL matrix notation, see Figure 5.) The matrix above could be said to be made up of ordered sets of row vectors, where each vector contains the food prices for a particular city.

Another common example is what is known as a coefficient matrix. Consider the following linear equation.

$$2x_1 + 4x_2 + 3x_3 = 8$$

Many times problems arise where a number of equations equal to the number of variables can be written (in this case, x_1, x_2, x_3 = 3 variables). It is then desirable to solve the set simultaneously for the values of x_1, x_2, and x_3 that satisfy all the equations. This process is known as solving simultaneous linear equations.

Such a set of equations might appear as

$$2x_1 + 4x_2 + 3x_3 = 8$$

$$10x_1 + 2x_2 + 4x_3 = 19$$

$$x_1 + 3x_2 - 5x_3 = 11$$

The digits directly to the left of each variable are known as coefficients. Most methods of solution require testing and manipulation of a matrix called the coefficient matrix. The coefficient matrix for the set of equations shown above would be

$$\begin{vmatrix} 2 & 4 & 3 \\ 10 & 2 & 4 \\ 1 & 3 & -5 \end{vmatrix}$$

In this example, the row vectors contain the coefficients associated with a particular equation. The column vector making up the matrix contains the coefficients associated with a particular variable. (For example, column one contains the coefficients associated with x_1.)

A matrix having m rows and n columns is said to be of the order m by n (m x n). The number of rows is the row order, while the number of columns is the column order. In Figure 5, the matrix $\overline{P}$ is of the order 5 by 5.

	1	2	3	4	5	Customary Notation	CAL Notation
1		X				$P_{1,2}$	P(1, 2)
2			X			$P_{2,3}$	P(2, 3)
3							
4							
5					X	$P_{5,5}$	P(5, 5)

Figure 5. $\overline{P}$ Array and CAL Notation in Two Dimensions

In principle, there is no limit to the number of dimensions which an array may have, provided that each array element is consistently supplied with one identifying subscript for each dimension of the array. In practice, many languages limit the number of subscripts to three, depending on the computer or processor used. Since CAL variables may have any number of subscripts, the number of dimensions an array may have is unlimited in the CAL language.

The following example will illustrate how subscripts composed of constants, variables, and arithmetic expressions are evaluated in CAL. We begin by letting the following variables contain the value indicated.

I = 3	R(1) = 2
J = 4	R(2) = 3
K = 2	R(3) = 6
L = -3	R(4) = 7
M = 5	R(5) = 8

Then, the subscripted variable P, in the two cases that follow, would be evaluated as shown below.

Example 1: P(R(I) - R(K), 2*R(K) + 2*L)

Substituting the values of the variables specified above, the first variable is evaluated as

R(I) - R(K) = R(3) - R(2) = 6 - 3 = 3

Following the same procedure for the second variable

2R(K) + 2L = 2R(2) + 2(-3) = 2(3) + (-6) = 6 - 6 = 0

Thus, the evaluated variable is P(3,0).

Example 2: P(R(J + L), R(R(J) - J), R(M) - I)

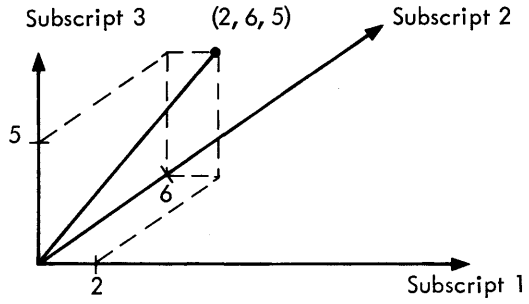
The three subscripts are evaluated as

R(J + L) = R(4 + (-3)) = R(4 - 3) = R(1) = 2

R(R(J) - J) = R(R(4) - 4) = R(7 - 4) = R(3) = 6

R(M) - I = R(5) - 3 = 8 - 3 = 5

Thus, the evaluated variable is P(2, 6, 5). Graphically, the subscript P(2, 6, 5) would be



EXPRESSIONS

Expressions are formed by combining operands with operators. Operands may be variables, functions, or constants. The operators may be arithmetic Boolean (logical), or conditional. The following are examples of expressions.

A + B - 2

A + B < C*D

A < B or C > D

LOG (A) = X*Y

X↑2 > SQRT (A ↑ 3)

ARITHMETIC OPERATIONS

Five arithmetic operations involving two operands (arguments) are used in CAL. The arithmetic operator symbols are given below.

Symbol	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
↑	Exponentiation (the process of raising a number to a power)

For example:

A+B	means	A plus B
A-C	means	A minus C
A*Q	means	A multiplied by Q
C/D	means	C divided by D
Y↑X	means	Y to the X power, Y^X
A↑2	means	A^2

In a limited sense, unary arithmetic operations, (i.e., operations involving only one argument or operand) are also possible in CAL. For example, the unary "minus" is available, but the quantity and its unary operator must normally be enclosed in parentheses, e.g., A + (-B), but not A + -B. If, however, the unary operator applies to the leading term of an expression, parentheses are unnecessary. For example, -A means the negative of A, +A means A, -A↑2 means the negative of A^2.

INTERPRETING THE ORDER OF COMPUTATION (PRECEDENCE)

The sequence in which the individual terms of an expression are to be evaluated and collected is unique. A precedence value is assigned for each of the arithmetic operations in CAL. The operations are listed below in descending order of precedence.

Symbol	Operation
↑	Exponentiation
-	Negation (unary)
*, /	Multiplication, Division
+, -	Addition, Subtraction (binary operations)

PRECEDENCE RULE – The precedence rule may be stated as follows: When unaltered by parentheses, the order of arithmetic operations performed within one expression is in descending order of precedence. Note, however, that the * and the / have the same level of precedence, as do the + and the -. When two or more operations of the same level of precedence appear in an expression, these operations are performed in order from left to right. Figure 6 shows several CAL expressions and their equivalent mathematical operations.

PARENTHESES IN CAL EXPRESSIONS

As used in the usual algebraic sense, parentheses may be introduced to override the rules of precedence for a given expression. This is frequently done for the purpose of simplifying mathematical expressions. In general, when a portion of an expression is enclosed in parentheses, the effect is to form a subexpression. The parentheses rule states that any subexpression must be evaluated before it can be employed in the rest of the main expression. Within any subexpression, the precedence rules given above apply. Examples of CAL expressions using parentheses are given in Figure 7. As a further example, consider the statement Q = A*(B+C*D), which is interpreted step by step as follows:

SET Q = A*(B+C*D)

Form R_1 as C · D

A*(B + R_1)

Form R_2 as B + R_1

A*R_2

Form R_3 as A · R_2 and assign to Q

Q = R_3

EXECUTING SINGLE COMMANDS

CAL, in combination with a remote terminal, can be used as an extermely powerful desk calculator to evaluate complicated mathematical expressions and to reduce data that requires involved calculations.

THE "TYPE" COMMAND

To use the terminal and CAL as a desk calculator, simply type the word TYPE, followed by the desired arithmetic expression. Then press the carriage return. CAL evaluates the expression and prints out the answer, observing all the rules of subscription, operator precedence, and parentheses stated above. Two examples are given below.

```
> TYPE          2.2543↑2*8.73 (RET)
2.2543↑2*8.73 = 44.364712

> TYPE          2+3*9/2.25+2↑4 (RET)
2+3*9/2.25+2↑4 = 30
```

Several expressions can be evaluated by one TYPE command, by separating the expressions with commas. If the series of expressions requires more than one line on the teletype paper, depress the line feed key at the end of each line for a continuation line. An example is given below.

```
> TYPE  2+3, 4-2, 2*3↑2, 3*2↑3/4 (LF)
2↑5, 3*6*2, (RET)
2+3=5
4-2=2
2*3↑2=18
3*2↑3/4=6
2↑5=32
3*6*2=36
```

In typing a CAL command, spaces can be used quite freely. The only exceptions are that no spaces can appear within a word, a number, or a variable name. Thus the expressions TYPE 2 + 3, 2 -3, 2 * 3, and 2↑3, are valid, but TYPE 1 000 000 is not. Note that commas are never used in typing numbers. For example, 1,000,000 must be typed as 1000000.

CAL Expression	Mathematical Expression
SET Q = A+(B/C↑F)	$Q = A + \frac{B}{C^F}$
SET Q = A/B*C	$Q = \frac{A}{B} \cdot C$ not $Q = \frac{A}{B \cdot C}$
SET Q = A+B/C+D↑E*F-G	$Q = A + \frac{B}{C} + D^E * F - G$
SET Q = A*B/C*D/E*F	$Q = \left(\frac{(AB/C)D}{E}\right) \cdot F$ or $Q = \frac{A \cdot B \cdot D \cdot F}{C \cdot E}$
SET Q = A*X↑3+B*X↑2/C	$Q = AX^3 + \frac{BX^2}{C}$
SET Q = C(K)+(A↑3)*B(J)/(9.7+3.5*P)	$Q = C_K + \frac{A^3 * B_j}{9.7+3.5P}$

Figure 6. Illustrations of the Mathematical Operator Precedence Sequence

CAL Expression	Mathematical Expression
SET Q = A+(B+C-(D*E+F))	Q = A+(B+C-(D·E+F)
SET Q = A*B-C/D+E*F	$Q = A \cdot B - \frac{C}{D} + E \cdot F$
SET Q = A*(B-C)/(D+E)*F	$Q = \left(\frac{A(B-C)}{D+E}\right) \cdot F$
SET Q = (C(K)+A(3)*B(J))/9.7+3.5*P	$Q = \frac{C_K + A_3 B_J}{9.7 + 3.5P}$
SET Q = A*B*C/(D*E*F)	$Q = \frac{A \cdot B \cdot C}{D \cdot E \cdot F}$
SET Q = (A*B*C)/(D*E*F)	$Q = \frac{A \cdot B \cdot C}{D \cdot E \cdot F}$
SET Q = ((A*B*C)/(D*E*F))	$Q = \frac{A \cdot B \cdot C}{D \cdot E \cdot F}$
SET Q = C + (A*X +B)*X	$Q = AX^2 + BX + C$
SET Q = 1/SQRT (M-4.5*N)	$Q = \frac{1}{\sqrt{M-4.5N}}$

Figure 7. CAL Expressions Using Parentheses and Mathematical Operator Precedence

The TYPE command may also be used to insert text by enclosing the text to be printed out in quotes: "xxxx", where xxxx is any text. Remember that continuation of lines may be accomplished by typing a line feed (i.e., the (LF) key) at the end of each line. This causes CAL to generate a carriage return and is otherwise completely ignored, except by the Q^c operation. An example is given below.

```
>TYPE "LINE FEED (LF) - IS THE CONTINUATION (LF)
CHARACTER WHICH CAUSES CAL TO GENERATE A (LF)
CARRIAGE RETURN AND PLACES THE TEXT (LF)
FOLLOWING EACH (LF) ON THE NEXT LINE" (RET)

LINE FEED (LF) - IS THE CONTINUATION
CHARACTER WHICH CAUSES CAL TO GENERATE A
CARRIAGE RETURN AND PLACES THE TEXT
FOLLOWING EACH (LF) ON THE NEXT LINE
```

ERRORS

The user may occasionally make typographical errors while typing a CAL command. CAL provides three ways of dealing immediately with such errors. Whenever CAL is accepting a statement, it will recognize the control characters A^c, W^c, and Q^c as editing characters. The notation of superscript c, as exampled above in A^c, W^c, and Q^c means that the user depressed the indicated alphabetic key (in this case A, W, or Q) and the control key simultaneously.

A^c

Control A prints a ↑ and deletes the immediately preceding character. It may be repeated to delete more than one character at a time. The correct character(s) may then be typed, as in the example given below.

```
>TYP 2+↑↑↑E 2+3
2+3=5
```

The command was interpreted correctly as TYPE 2+3. Note that there were three characters deleted (A^c, A^c, A^c) to remove the +, and 2, and the blank, in that order.

W^c

Control W prints a \ and causes the last word typed to be deleted. More precisely, it causes all immediately preceding blanks to be deleted, then the word, and then all blanks before the word up to the last word (variable, expression, or number) typed. The correct word may then be typed, as in the example below.

```
>TIPA \ TYPE 2-3 \ 2+3
2+3 = 5
```

It is important to remember that spaces are important in this edit, since by deleting 2-3 the spaces preceding and following it are also deleted. In retyping the correct word, the user would have to type the appropriate space.

Q^c

Control Q prints an ← and causes the entire typed line to be deleted. It must be typed before a carriage return and line feed are performed. It then deletes all the characters on the current line, back to the beginning of the line. In the case of a continuation text line, it deletes back to the line feed only, not back to the beginning of the actual line. The correct line may then be typed. For example

```
>TYPE "IN XANADU DID KUBLAI KHAN (LF)
A STATELY PLEASURE DOME DECREE (LF)
WHERE ALPH, THE SACRED RIVER, RAN (LF)
THROUGH CAVERNS MEASURELESS TO MAN (LF)
NOW IS THE TIME FOR ALL GOOD MEN← (LF)
DOWN TO A SUNLESS SEA."
```

At execution, this will read

```
IN XANADU DID KUBLAI KHAN
A STATELY PLEASURE DOME DECREE
WHERE ALPH, THE SACRED RIVER, RAN
THROUGH CAVERNS MEASURELESS TO MAN
DOWN TO A SUNLESS SEA.
```

CAL accepts a number of other control characters. Their function is described in Chapter 3.

ERROR DIAGNOSTICS

A second type of error occurs when the user gives an incorrect command or fails to detect a typographical error. When CAL detects an error of this sort, it types out a comment intended to help the user locate the problem, as in the example given below.

```
>TYPE (5+A+6)/(8+8) (RET)
ERROR ABOVE: A NEVER SET

>TYPE (5+5+6)*(6-2+3 (RET)
CHECK ( )
```

(Note that the error is in the omission of a right parenthesis.)

```
>TYPE (5+5+6)*(6-2+3)) (RET)
JUNK ON END
```

(The error is in the superfluous parenthesis at the end of the expression.)

```
>TIPE 6 + 5 (RET)
?
```

When commands are misspelled, all that appears is a question mark.

These comments from the computer are quite useful for the beginner who is learning to use CAL. They provide a means of learning by doing.

THE "SET" COMMAND

CAL utilizes the command SET to perform arithmetic assignments. Arithmetic assignment statements employ the replacement operator, =, to define numerical quantities. For example, consider the expressions:

```
>SET A(J) = 2
>SET J = J + 1
>SET C(I) = A(I)*2+B(I)↑2
```

The assignment operator "=" assigns numerical values to A(J), J and C(I) respectively by evaluating the numerical value of the expressions to the right of the signs. The "=" means "the variable on the left is replaced by the value of the variable or expression to the right." The CAL command for performing arithmetic assignment is the SET command, but the word "SET" may be omitted. Thus, SET Z = A(J) means that value of Z is replaced by the value of A(J). The general form of the SET command is

SET v = e or v = e

which means replace the current value of any variable, v, with the current value of an arithmetic expression, e.

```
> SET A = 2+3+4
> SET A(1) = 7.95 ↑10.66*3.47/2*6.8 ↑ 4.23 (RET)
```

Once a variable has been defined, it can be used to define subsequent variables, or its value can be printed out by using the TYPE command.

```
> SET A=2+3+4
> SET A(1)=7.95↑10.66*3.47/2*6.8↑4.23
> SET X=2+3
> SET A(2)=2↑4
> SET B=3*5
> SET C=A+B
> SET D=(X/A(1)+3)↑2
> TYPE A, A(1), X, A(2), B, C, D
```

The computer prints out

```
   A = 9
A(1) = 2.2841561     13
A(2) = 16
A(1) = 2.2841561     13
   X = 5
A(2) = 16
   B = 15
   C = 24
   D = 9
>
```

In the above example, note that the first portion of the TYPE command to be executed was TYPE A. If A is a subscripted variable, all values of A that have been set prior to the issuance of the command will be typed. This accounts for the order and duplication above. If a subscripted value is included such as A(1) or A(2), only that array value will be typed. Summarizing, in the above example, TYPE A produced

```
   A = 9
A(1) = 2.2841561     13
A(2) = 16
```

TYPE , A(1) produced

```
A(1) = 2.2841561     13
```

TYPE , , X, produced

```
X = 5
```

No other X elements such as X(1), X(2), etc., existed. The same is true for B, C and D.

TYPE , , , A(2), produced

```
A(2) = 16
```

The value of a variable may be changed at any time by giving it a new value with a new SET command.

```
> TYPE A
A = 9
> SET A = 2+5 (RET)
> TYPE A (RET)
A = 7
```

When a variable has been given a new value, CAL "forgets" the original value. Occasionally, it is desirable to eliminate a variable altogether without giving it a new value. This is accomplished by using the DELETE command.

```
> DELETE A (RET)
> TYPE A (RET)

ERROR ABOVE: A IS UNDEFINED
(an error comment by CAL)
```

Further use of the DELETE command is discussed in Chapter 3.

MATHEMATICAL FUNCTIONS

In addition to the five arithmetic operators described previously, a number of mathematical functions are also available in CAL. These are listed below.

ABS(A)	Absolute value of A[t]
SIN (A)	Sine of A[t]
COS (A)	Cosine of A
TAN (A)	Tangent of A
ATAN (X, Y)	Arc Tangent (X/Y) ($\pi > R > -\pi$)
EXP(A)	E to the power A
LOG (A)	Natural logarithm
LOG10 (A)	Base 10 logarithm
SQRT (A)	Positive square root of A
IP (B)	Integer part of B (If B = 246.25, IP = 246)
FP (B)	Fractional part of B (If B = 246.25, FP = .25)
SUM (A, B, C, D, . . .)	Sums the list
PROD (A, B, C, . . .)	Finds the product of the list
MAX (A, B, C, . . .)	Finds the maximum in the list
MIN (X, Y, Z, A, B, . . .)	Finds the minimum in the list

[t]The trigonometric functions take their arguments in radians or return results in radians.

The arguments of these functions may be any of the expressions defined previously, including the functions themselves. The rules of precedence for the arithmetic operators and parentheses will be observed in evaluating the arguments. Any meaningful combination of these functions and the operators described earlier can be used.

```
> TYPE SIN(2) + COS (0)*SQRT(4)/LOG(10) (RET)
SIN(2) + COS(0)*SQRT(4)/LOG(10) = 17778864
```

```
>TYPE LOG (20)+MAX(A, B, C, D, E) (RET)
```

(Note: Assuming A, B, C, D, and E have been defined)

```
LOG(20)+MAX(A, B, C, D, E) = 25. 30103
```

```
> TYPE LOG(3+(4↑.76)) (RET)
LOG(3+4↑.76)) = 1.7694896
```

```
> TYPE LOG(COS(SIN(SQRT(900)*PI/180))) (RET)
LOG(COS(SIN(SQRT(900)*PI/180))) = -0. 13058484
```

Since trigonometric functions in CAL require that angles be expressed in radians, the numerical constant π is frequently useful in working with these functions. CAL has a special variable called PI which contains the value of π to 7 places (3. 1415927). This may be called at any time.

```
> TYPE SIN (PI/2), SIN(2*PI) (RET)
SIN(PI/2) = 1
SIN(2*PI) = 0. 9280387-07
```

Note: 2π radians = 360 degrees. Therefore, if the sine of 30 degrees is desired, the statement is

```
> TYPE SIN(30*PI/180) (RET)
SIN(30*PI/180) = . 5000000
```

Additional examples of the CAL functions are included in the chapters that follow.

Also available in CAL is the MOD function, which is written in the format

```
e MOD m
```

where e is any expression and m is any number.

MOD stands for MODULO ARITHMETIC. Without giving lengthy explanations, the "result" of a modulo problem is the remainder of a division operation.

For example, to calculate 254 MODULO 8, divide 254 by 8. The remainder is the answer. This can be expressed in CAL by

```
> TYPE 254 MOD 8
254 MOD 8 = 6
```

Other examples are

```
> SET X = 5
> TYPE X MOD 2
X MOD 2 = 1
> TYPE 3 + 3 MOD 2
3 + 3 MOD 2 = 4
```

Note that by using MOD 2, all odd numbers result in a 1 and all even numbers result in a 0.

```
> TYPE (3 + 3) MOD 2
(3 + 3) MOD 2 = 0
```

3. CAL PROGRAMS

In addition to executing statements one at a time, as discussed in Chapter 2, CAL can be used to store commands for future execution. This is done through a CAL program composed of a series of statements called steps, each of which begins with a step number and includes a specific command to the computer. (A few direct commands never require a step number. These are discussed in a later section.)

To write a CAL program, the user will need to know how to write, add, delete, and edit steps, how to use the commands that are available, and how to use the logical operators and special clauses. The purpose of this chapter is to explain these things in detail, with appropriate examples.

STEPS AND PARTS

As stated above, each statement in a program begins with a number which has the form i.j. The allowable range of numbers for CAL statements is from .0001 to 999999.99999. The integer number formed by the digits to the left of (preceding) the dot is called the part number of the step. The entire number, i.j, is called the step number. There can be several steps in a part but each statement has its own unique step number. Steps may be referred to singularly by their individual numbers, or collectively by the part number. For example, consider the collection of statements below.

```
>4.0    SET        X = Y
>4.1    SET        X = X+1
>4.2    SET        Z = X*Y
>4.3    DONE IF    X-Y<0
>4.5    TYPE       X,Y
```

The command "DO STEP 4.2" would cause execution of STEP 4.2 only and return to the original point of the DO command. The command "DO PART 4.0" would cause execution of all the steps whose integer part (that number preceding the dot) is 4, in this case 4.0, 4.1, 4.2, 4.3, and 4.5. The DO command in the statements above is used at this time only to illustrate the idea of step numbers; the DO command is explained fully later in this chapter.

A linear ordering of statement numbers is defined by taking them as ordinary decimal numbers. Thus, $1 < 1.01 < 1.1 = 1.10 < 1.2 < 2 = 2.0 < 10.0$. The ordering of statements in the program is determined by their statement numbers and not by the order in which they are typed. If several statements with the same number are input, the last one typed will be kept.

Before going into a detailed discussion of programming in CAL, a sample program is illustrated in Figure 8 below. The example, which begins with a log in, computes the hypotenuse of a right triangle by the Pythagorean theorem.

```
PLEASE LOG IN!     E4 SDS; BEN (RET)
READY 11/28  18:41
-CAL (RET)
CAL
NUMBER OF STATEMENTS NEEDED = 10 (RET)
HEADING, PLEASE
HYPOTENUSE (RET)
     (seven lines will be skipped)
HYPOTENUSE PAGE 1 (RET)

>3.1   DEMAND X,Y (RET)
>3.2   SET Z = SQRT(X↑2+Y↑2) (RET)
>3.3   TYPE X,Y,Z (RET)
>3.5   TO STEP 3.1 (RET)

>TO STEP 3.1 (RET)          (This statement, without
                             a line number, is one way
                             to activate the program.)
     (At execution, the following occurs.)
X = 3
Y = 4
X = 3,  Y = 4,  Z = 5
X = (The values are again demanded. The program
     will continue until the ESC key on the Teletype
     is depressed.)
```

Figure 8. Example of a CAL Program

This example is typical of a short CAL program. Note that the integer part (the part left of the decimal) of all the steps is 3, indicating that all the steps belong to the same part. For example, step 3.3 is one of the steps in part 3. The five steps have now been stored by CAL. Together they form a program having one part (part 3.0) and four steps (steps 3.1, 3.2, 3.3 and 3.5). The step without a statement number (TO STEP 3.1) will actuate the program once. This step will not, however, be stored. One way of re-executing the program is to retype the statement TO STEP 3.1, without a step number. In this example, the program continuously executes because of step 3.5.

ADDING OR DELETING STEPS

Suppose that in the example above, the desired result at the end of part 3 is to compute C, as shown in Figure 9.

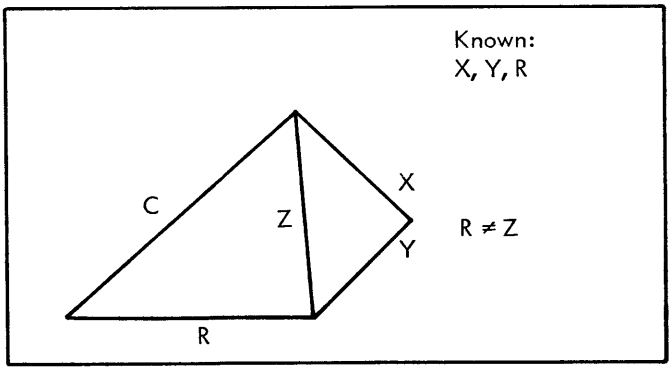


Figure 9. Hypotenuse Problem

The program may then be expanded by adding the following instructions.

```
> 3.12    DEMAND R (RET)
> 3.25    SET C = SQRT(R↑2+Z↑2) (RET)
> 3.40    TYPE R (RET)
```

Thus, at execution or on a relisting, the program will appear and execute as follows:

```
> 3.1     DEMAND X, Y (RET)
> 3.12    DEMAND R (RET)
> 3.2     SET Z = SQRT(X↑2+Y↑2) (RET)
> 3.25    SET C = SQRT(R↑2+Z↑2) (RET)
> 3.3     TYPE X, Y, Z (RET)
> 3.40    TYPE R (RET)
```

A second way of accomplishing the same thing is to take advantage of the fact that if a step is retyped with the same statement number, the last one entered will be saved. Therefore, type

```
> 3.1     DEMAND X, Y, R (RET)
> 3.25    SET C = SQRT(R↑2+Z↑2) (RET)
> 3.30    TYPE X, Y, Z, R (RET)
```

Then, at execution or relisting, the program would appear as follows.

```
> 3.1     DEMAND X, Y, R (RET)
> 3.2     SET Z = SQRT(X↑2+Y↑2) (RET)
> 3.25    SET C = SQRT(R↑2+Z↑2) (RET)
> 3.30    TYPE X, Y, Z, R (RET)
```

If the user wished to restore the program to its original form, this would be accomplished by the DELETE command.

```
> DELETE STEP 3.12 (RET)
> DELETE STEP 3.25 (RET)
> DELETE STEP 3.40 (RET)
```

At execution or relisting, the program would be restored to its original form.

CONTROL CHARACTERS

Although steps may be completely retyped to correct errors, there are easier ways to edit programs. If the error is contained in the line last typed and is immediately discerned by the user, he has the option of using several control characters to edit the line by deleting, changing, adding, or inserting characters. Although he may ignore these edit characters in favor of retyping lines, the user will find them very useful in editing a program once he understands their operation. Essentially, the control characters allow the user to edit a line without retyping whatever portion of the line is correct.

The control characters and their functions are explained below and examples given of each. For the purposes of this discussion, "original line" means the line last typed, i.e., the line being edited. The term "new line" means the line the computer prints out as the editing process progresses. At the end, the new line replaces the original line.

C^c

Control C causes a single character of the original line to be printed out on the new line. In itself, this operation does not change any character in the new line. It simply locates the user at the point on the line where he wishes to delete, add, etc. If the REPEAT key is depressed at the same time, the computer will repeat, character by character, until the REPEAT key is released. For example, consider the following line as typed by the user:

```
> 3.1  TYPE A = I FOR I = 1 BY 200 TO  N  (RET)
>
```

On reading this line over, the user realizes that it contains an error: that is, the integer 200 should be 2.

To edit immediately, the user presses the Control C and the REPEAT key simultaneously, stopping when he reaches the digit 2 in the integer 200. The computer then prints out the line, character by character, up to and including the 2.

```
> 3.1  TYPE A = I FOR I = 1 BY 2
```

The user is now in a position to delete the two zeros. This operation is explained below in the discussion of control X.

Z^cx

Control Z performs the same function as C^c, but it prints out more than one character at a time. This is accomplished as follows: The user selects any letter, number, punctuation, or operator symbol (as indicated by the x in Z^cx) that appears in the original line; he then types the Z^c followed by the character selected. This causes the original line to be printed out, up to and including that character. If the selected character does not appear in the original line, CAL rings a bell and takes no other action. Thus, the user in the above example could have typed a Z^c2 to effect a printout of the

line through the 2. The choice of which control character to use will usually depend on the location of the error to be corrected. In either case, the computer will print out the original line to the designated character and await further editing.

S^C

Control S is used to delete characters. Each use of the control character causes a single character of the original line to be deleted and a % to be printed in its place. (The % does not appear in the final version.) Thus, in the example cited, the user would strike the S^C twice to delete the two zeros, once he had arrived at their location by using the C^C or Z^C. The new line as it would then be printed out on the Teletype is:

```
>3.1 TYPE A = I FOR I = 1 BY 2%%
```

$X^C x$

Control X is also used to delete characters, but it will cause more than one character at a time to be deleted. The user in the example above might decide to simply eliminate the remainder of the line. In this case, he would type the X^C and an N (the last character in the line). The computer would then print out a % for each character and space in the remainder of the original line up to and including the N.

```
>3.1 TYPE A = I FOR I = 1 BY 2%%%%%%%%
```

Note that five additional symbols are printed out (the first two having been printed out by the S^C), because two blanks were in the remainder of the line, in addition to TO N, and the computer treats the blanks as characters. Having finished the editing of the line, the user could type a carriage return, which would cause the original carriage return and line feed to be deleted while signaling the computer that the program is to continue.

Any time the carriage return is typed, the remainder of the original line will be deleted and a line feed will be generated by the computer to continue the program. That is, the user will be out of the edit mode. To re-enter it, it is necessary to type the EDIT command (discussed later in this chapter).

E^C

Control E is used to insert new text in the original line. At the first use of the E^C, a "<" sign is printed, at which point the user types in his insertion. He then types a second E^C, which prints a ">" sign and tells the computer that he has completed inserting text. He may then go on with his editing.

As an example of the operation of control E, consider the following example.

```
> SET X = Y/Z
```

The user has erroneously indicated that Y should be divided by Z, whereas he wished to specify that Y should be multiplied by Z. His first step is to type Z^CY (to arrive at the point of deletion); he then types a control S to delete the sign in error. He is now ready to insert a multiplication sign (*). He types an E^C, an *, and a second E^C, in that order. The new line would appear as

```
> SET X = Y% <*>
```

The user may then go on with his editing. Note that, upon relisting, the symbols "%", "<", and ">" would not appear in the line.

R^C

Control R is used to permit recovery in cases where the user wishes to verify the edit thus far or where he has become uncertain concerning the state of the edit and wants to see his results printed out. R^C causes CAL to perform a line feed and a carriage return and then to reprint the remainder of the original line, after which it reprints on the next line the edited portion of the line. For example, the user has typed the first line below (the original line) and edited a portion of it (the new line). After deleting the "4*H", he strikes R^C, which results in a print out (third and fourth lines).

```
> SET E=5*G+4*H+3*I+2*J+K
> SET %<M>=5*G+%%% (RET)
                  +3*I+2*J+K
SET M=5*G+
```

T^C

Control T operates the same as control R, except that the original line is correctly aligned with the new one (although still on two lines). This operation takes longer for the computer to accomplish.

D^C

Control D prints the remainder of the original line onto the new line and then causes the new line to replace the original one in the program. The user may then continue writing his next step. For example, assume that the user has typed a statement with an error in it and has corrected that error (by inserting A(2) to A(3)), using the control techniques previously explained.

```
> 3.2 TYPE A(1), A(2), A(4), A(5) (RET)
> 3.2 TYPE A(1), A(2), <A(3), >
```

Typing a control D at this point causes the remainder of the original line to be printed on a new line, a carriage return and a line feed to be generated, and the edit mode to be discontinued.

```
>3.2 TYPE A(1), A(2), <A(3), >A(4), A(5) (RET)
```

The user then is ready to go on with his next step.

F^C

Control F performs exactly the same function as control D, except that it copies the remainder of the original line to the new one without printing it for the user to view.

Again, the edit mode is ended by this control character and the user continues on to his next step.

Both control D and control F may be used to execute the same direct statement more than once in a program. For example, assume that the original statement is

```
> TYPE LOG(A), A, SIN(B*PI/180) (RET)
```

If control D is typed by the operator the statement will reappear

```
> TYPE LOG(A), A, SIN(B*PI/180)
```

and be executed. This procedure could be repeated any number of times. If control F is used, the statement will not be reprinted, but it will be executed again.

Y^C

Control Y also copies and prints the remainder of the original line to the new one, but it does not end the edit mode. The user may continue to add to the line. Thus, in the example given for control D, the user might wish to add A(6) to the new line. He would then simply type in A(6) directly. Once he has completed editing the line, he types a carriage return to signal the computer that he wishes to continue with his program.

THE "EDIT" COMMAND

If a statement is to be edited at a time later than directly after it was typed, it must first be recalled by the EDIT command. The user types:

```
EDIT STEP i.j (RET)   (where i.j is the step number)
```

The specified step is then printed out by the computer and, in effect, becomes the original line for the edit operations using the control characters presented above. The statement is not deleted or altered by the EDIT command but is simply printed out ready for editing. All of the EDIT control characters may be used once the EDIT command is invoked. The following variations to the command are also possible:

```
EDIT FORM e      (where e may be any expression)
EDIT f           (where f may be a function only)
```

"FOR" AND "DO"

It has been shown that some rather complex numerical evaluations can be performed with CAL on a remote terminal. It should be apparent, however, that extensive typing may be required to get a single answer. In the example given in Figure 8, it is often desirable to calculate Z for several values of X and Y without re-cycling or re-initiating the program. This can be accomplished by using the FOR clause and DO command.

THE "FOR" CLAUSE

Single commands or simple programs can be augmented for modification with a FOR clause. Briefly, a FOR clause causes the command in question to be executed repeatedly for a specified set of values of some variable. In the following example

```
> TYPE LOG(A) FOR A=1, 5, 10 (RET)

LOG(A) =  0
LOG(A) =  1.6094379
LOG(A) =  2.3025851
```

the FOR clause first causes A to be set to 1 and then causes the TYPE command to be executed. Next it repeats the same procedure, setting A to 5 and then to 10. In the example above, each value to which the FOR variable was to be set was specified. The user can also specify that the variable be set to some beginning value and increased or decreased repeatedly by a specified amount until some final value is reached.

```
> TYPE 2↑A FOR A=2 TO 5 (RET)

2↑A =  4
2↑A =  8
2↑A =  16
2↑A =  32
```

Here, A was SET to the first number after the FOR (2), after which the TYPE command was executed. Next, A was increased by one and TYPE was executed again. This procedure was repeated until the final value (5) was reached. If incrementation is other than by the number one, it must be specified as follows.

```
> TYPE 2↑A FOR A=2 BY 2 TO 10 (RET)

2↑A =  4
2↑A =  16
2↑A =  64
2↑A =  256
2↑A =  1024
```

Whenever the BY is omitted from the FOR clause, CAL increments by one.

Both ways of specifying FOR values can be combined in a single FOR clause.

```
> TYPE 2*A FOR A=1, 10, 19 BY .5 TO 20, 25 (RET)

2*A =  2
2*A =  20
2*A =  38
2*A =  39
2*A =  40
2*A =  50
```

The FOR clause provides a conveinent way of calculating tables of numerical values. For example, the following command produces a table of squares, cubes, square roots, and cube roots.

TYPE A, A↑2, A↑3, SQRT(A), A↑(1/3) FOR A=1 TO 3 ⏎

```
      A =  1
    A↑2 =  1
    A↑3 =  1
SQRT(A) =  1
A↑(1/3) =  1
      A =  2
    A↑2 =  4
    A↑3 =  8
SQRT(A) =  1.4142136
A↑(1/3) =  1.2599210
      A =  3
    A↑2 =  9
    A↑3 =  27
SQRT(A) =  1.7320508
A↑(1/3) =  1.4422496
```

In the examples above, the values to which the FOR variables were to be set were specified by numbers. These values can also be specified by expressions, provided CAL is able to calculate the values of the expressions used. For example, one could type

> TYPE A FOR A=1 BY LOG(8.7*10.1/2+3) TO 22.13 ⏎

```
A =  1
A =  4.8487637
A =  8.6975273
A =  12.546291
A =  16.395055
A =  20.243818
```

Note that if the FOR variable never precisely equals the final value specified, the FOR clause terminates at the last value before the final value was exceeded.

Finally, two or more FOR clauses may be used to modify a single command. For example, to calculate the area of a set of ellipses (where A and B are the semi-axes) one could type

> TYPE A, B, PI*A*B FOR B=1 TO 4 FOR A=1 TO 3 ⏎

```
    A =  1
    B =  1
PI*A*B =  3.1415927
    A =  1
    B =  2
PI*A*B =  6.2831853
    A =  1
    B =  3
PI*A*B =  9.4247780
    A =  1
    B =  4
PI*A*B =  12.566371
    A =  2
    B =  1
PI*A*B =  6.2831853
    A =  2
    B =  2
PI*A*B =  12.566371
    A =  2
    B =  3
PI*A*B =  18.849556
    A =  2
    B =  4
PI*A*B =  25.132741
    A =  3
    B =  1
PI*A*B =  9.4247780
    A =  3
    B =  2
PI*A*B =  18.849556
    A =  3
    B =  3
PI*A*B =  28.274334
    A =  3
    B =  4
PI*A*B =  37.699112
```

In this case, CAL sets the rightmost FOR variable to its initial value, and then steps through all values of the next left FOR variable. Next, the rightmost FOR variable is incremented, and again CAL steps through all values of the next left FOR variable. This procedure is repeated until the rightmost FOR variable has been stepped through all its values.

THE "DO" COMMAND

To apply the same principle to the example given in Figure 9 (i.e., to execute part 3 for several values of X and Y), the DO command is used, as shown below.

```
> 2.0 DO PART 3 FOR I=1 TO N ⏎
> 2.1 PAUSE ⏎
> 3.2 SET Z(I) = SQRT(X(I)↑2 + Y(I)↑2) ⏎
> 3.3 TYPE X(I), Y(I), Z(I) ⏎
```

In executing the DO command, CAL "did" (executed) each step in numerical order in the part referred to by DO. In the example just given, the DO command is combined with a FOR clause. In this case, each step is performed in numeric order for each value of I, in increments of one until I exceeds N. CAL then returns to the next highest step in part 2. Step 2.1 in the example above causes the program to stop for new instructions from the operator. If several DO commands are executed according to sequential parts of the program, powerful programming techniques can be built from the combinations of DO commands appended by FOR clauses.

In computer programming terminology, the DO command allows CAL to execute a part of a program as a subprogram

within the main program. In a long program, a single part could be executed repeatedly at different points in the program by referring to it repeatedly with separate DO commands. An example of using the DO command as a subprogram is shown in Chapter 4.

DO commands can refer to steps as well as parts in a program. In this case, the step in question is executed and then CAL returns to the command following the DO. For example, the program to compute several values of the hypotenuse of a right triangle could be written as follows.

EXAMPLE OF A HYPOTENUSE PROGRAM USING THE "DO STEP" COMMAND

```
>1.0 DO STEP 1.2 FOR I=1 TO N (RET)
>1.1 TO STEP 1.3 (RET)
>1.2 SET Z(I) = SQRT (X(I)↑2 + Y(I)↑2) (RET)
>1.3 DO STEP 1.5 FOR I=1 TO N (RET)
>1.4 PAUSE (RET)
>1.5 TYPE I, Z(I) (RET)
```

Note that when a DO STEP is used instead of a DO PART, a transfer statement (step 1.1) must be inserted after the DO command. After the DO command is executed for I=1 in increments of one (1) to I=N, the program will execute the next sequential step in the program. If the transfer step were omitted, step 1.2 would be executed one additional time for I=N+1.[†] The transfer step carries execution around step 1.2 when the FOR clause is satisfied. Although part numbers are denoted by integers, a DO command can refer to a part number having a fractional part, e.g., part 5.61 or part 4.8. In this case, CAL will execute steps in the part in question, beginning at the step indicated by the number in question. For example, DO PART 4.56 would cause CAL to do all steps in part 4 from step 4.56 on. If there were no step 4.56, the next step greater than 4.56 in part 4 would be executed first.

Another example that illustrates the use of the DO and FOR commands is a program that calculates mean and standard deviations, given N numbers. The formulas used are

$$\text{Mean} = \frac{\Sigma x_i}{N}$$

$$\text{Standard deviation} = \sqrt{\frac{\Sigma(x_i)^2 - \frac{(\Sigma x_i)^2}{N}}{N-1}}$$

```
>0.5 DEMAND N (RET)
>1.0 SET S = 0 (RET)
>2.0 SET Q = 0 (RET)
>3.0 DO PART 5.0 FOR I = 1 TO N (RET)
>4.0 TO STEP 6.0 (RET)
>5.0 DEMAND A(I) (RET)
>5.2 SET S = S + A(I) (RET)
>5.4 SET Q = Q + A(I) * A(I) (RET)
>6.0 TYPE S/N (RET)
>7.0 SET D = SQRT (Q - ((S *S)/N) /N -1) (RET)
>8.0 TYPE D (RET)
```

Note that all the steps of part 5 are executed N times, and that step 4.0 prevents another pass through part 5.

DECISION COMMANDS AND COMPARISON OPERATORS

THE "IF" CLAUSE

In executing a program, CAL can make decisions as to what action to take depending on the current values of variables or expressions within the program. Decisions are specified with the IF clause. Like the FOR clause, an IF clause modifies a CAL command such as TYPE, SET, DO, or TO STEP.

For example,

```
1 DO PART 2 FOR I=1 BY 7 TO 15

2 SET A=I
2.1 TYPE A IF A<10
```

will produce

```
A = 1
A = 8
```

or any other value for A less than 10. If A is 10 or greater, execution will not take place.

Note that the command

```
>TYPE "THE VALUE OF 'A' IS LESS THAN 10" IF A < 10
```

is an illegal instruction, because a TYPE literal may not be qualified.

An example using the SET command would appear as follows.

```
>1.0 DO PART 2.0 FOR A = 1 TO 10 (RET)
>2.0 SET C = 0 IF A↑2 < 50 (RET)
>2.1 SET C = 99 IF A↑2 > 50 (RET)
>2.2 TYPE A, C (RET)
> TO STEP 1.0 (RET)
```

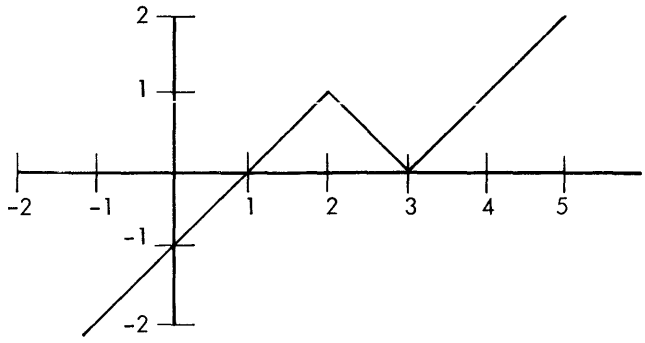
At execution

A = 1	C = 0
A = 2	C = 0
A = 3	C = 0
A = 4	C = 0
A = 5	C = 0
A = 6	C = 0
A = 7	C = 0
A = 8	C = 99
A = 9	C = 99
A = 10	C = 99

[†] In the FOR clause, the test is for I > N, so that I is incremented and then tested. Transfer out of the DO command would occur at I = N + 1.

Example of a Graph of Y = f(x)

As a more comprehensive example, consider a program to compute Y, whose graph is shown below.



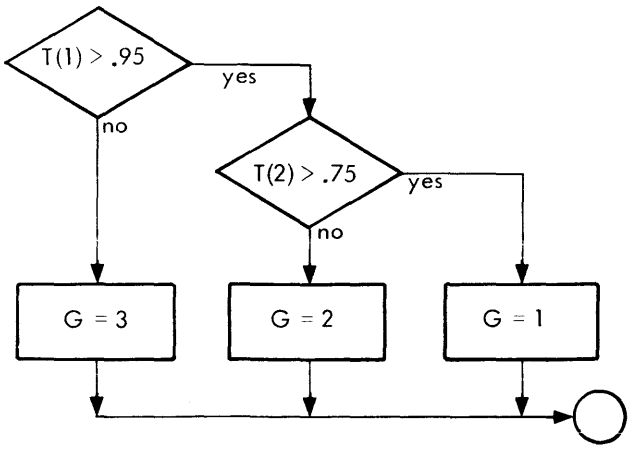

```
> 1.0 DEMAND X (RET)
> 1.1 SET Y = X-3 IF X > = 3 (RET)
> 1.2 SET Y = 3-X IF X > 2 AND X < 3 (RET)
> 1.4 SET Y = X-1 IF X < = 2 (RET)
> 1.5 TYPE  Y (RET)
> TO STEP 1.0
```

At execution the computer will print, as a result of step 1.0, X = -2 (user enters the value of -2).

For X = -2, the IF clause in step 1.1 cannot be satisfied since X is less than 3. Execution then passes to step 1.2 which again cannot be satisfied because of the IF clause. Therefore, execution passes to step 1.4. Since -2 is less than 2, the IF clause is satisfied and Y is set to (-2 -1) = -3. Note that in each IF clause a comparison operator was needed. The comparison operators are discussed immediately preceeding the examples in the next section.

Example of a Test for the Quality of Steel

Consider, as a further example, a test for the quality of steel. Steel is considered Grade 1 if Test One exceeds .95 and Test Two exceeds .75; Grade 2 if Test One exceeds .95 but Test Two does not exceed .75; and Grade 3 if Test One does not exceed .95. The flow diagram and program would be as shown below.




```
> 1.0 DEMAND T(1), T(2) (RET)
> 1.1 TO STEP 1.7 IF T(1) < .95 (RET)
> 1.2 TO STEP 1.5 IF T(2) < .75 (RET)
> 1.3 SET G = 1 (RET)
> 1.4 TO STEP 1.8 (RET)
> 1.5 SET G = 2 (RET)
> 1.6 TO STEP 1.8 (RET)
> 1.7 SET G = 3 (RET)
> 1.8 TYPE G (RET)
```

COMPARISON OPERATORS

In general, the decision in an IF clause is based on a comparison between two expressions, variables, or numbers, an expression and a number or variable, etc. In the preceding examples, an input variable was compared with a number by means of the "greater than", the "greater than or equal to", and the "less than or equal to" comparison operators. In CAL there are six comparison operators as follows:

=	equals
#	not equal
<	less than
>	greater than
< =	less than or equals
> =	greater than or equals

Note that when the "=" sign is used as a comparison operator it has the same meaning we understand in mathematics (as opposed to its use with SET where it denoted replacement). For example:

```
SET Y = 5 IF X = 3
```

means replace the current value of Y with 5 if the current value of X is equal to 3. The first "=" sign is a replacement, the second is a comparison operator.

Example of a Sales Tax Problem

The following program to compute the sales tax for a particular purchase is a further example of the use of the IF clause, comparison operators, and a CAL subroutine to find the integer and floating point portions of numbers. For amounts over one dollar, the tax is the tax on the dollar amount plus the rate shown for the fractional part of a dollar.

Assumed tax rates

0-15 ¢	0 ¢
16-37 ¢	1 ¢
38-62 ¢	2 ¢
63-84 ¢	3 ¢
85-100 ¢	4 ¢

```
> 1.0 DEMAND C (RET)
> 2.0 SET T = .04 * IP (C) (RET)
> 2.1 SET T = T+ .01 IF FP (C) > .15 (RET)
> 2.2 SET T = T+ .01 IF FP (C) > .37 (RET)
> 2.3 SET T = T+ .01 IF FP (C) > .62 (RET)
> 2.4 SET T = T+ .01 IF FP (C) > .84 (RET)
> 3.0 TYPE T, T+C (RET)
> 4.0 TO STEP 1.0 (RET)
```

BOOLEAN (LOGICAL) OPERATORS

In most of the foregoing examples, the general structure of the IF clause was "IF" followed by a comparison sub-clause based on a comparison operator. The power of the IF clause can be expanded by using another kind of operator, the Boolean operator, to modify or link together comparison sub-clauses. The Boolean operators available in CAL are "AND", "OR" and "NOT".

```
> DO PART 4 IF (A = B AND C = D) OR (LF)
  (E = F) NOT (G > H) (RET)
```

PART 4 can be executed under either of the following conditions:

(1) A = B, C = D, G < H

(2) E = F, G < H

```
>DO PART 4 IF A = B AND NOT (E = F AND G > H)
```

PART 4 will be executed if

(1) A = B

(2) E ≠ F

(3) G ≤ H

Note that, in the examples above, parentheses were used to indicate the scope of the Boolean operators in the same way as in an algebraic expression and that Boolean operators can be combined (e.g., AND NOT).

The following will illustrate the use of IF clauses with comparison and Boolean operators.

PRECEDENCE IN THE BOOLEAN OPERATORS

In the general scheme of computational precedence, the Boolean operators have the lowest precedence. The total order of precedence is:

The five arithmetic operators

The six comparison operators

The three Boolean operators whose relative precedence is NOT, AND, OR

The proper use of precedence permits the user to eliminate some steps in writing an expression. For example

```
A AND B = C+D*E↑F
```

means the same as

```
A AND (B = (C+(D*E↑F)))
```

NEGATIVE NUMBERS WITH BOOLEAN OPERATORS

Since negative numbers in Boolean operators are always considered true, care must be taken in using them. Otherwise, illogical results may be produced. As an example, consider the following program step.

```
> 3.1 SET A = 3 IF X OR Y = Z
```

If X = -1, Y = 2, and Z = 1, then at execution the logical test Y = Z would yield a zero (i.e., Y does not equal Z). The OR operator then adds the expression preceding it to the expression following it. The result is -1 +0 = -1, which indicates that the condition is true. This means that A would then be set to 3. However, this result is obviously undesirable since neither X nor Y = Z in this case. A flow diagram of this clause would be as follows:

```
IF X OR Y = Z

X      Y      Z
-1     2      1
        \    /
         \  /
-1  +     0   =    -1 (indicates true)
```

To obtain the correct result, the step should be written as

```
> 3.1 SET A = 3 IF X = Z OR Y = Z
```

Then the clause would be executed correctly, as shown in the flow diagram below.

```
IF X = Z    OR    Y = Z

-1      1     2      1
 X  =   Z     Y  =   Z
  \    /       \    /
   \  /         \  /
    0     +      0     =   0 (indicates false)
```

The same kind of illogical results can occur if negative numbers are used in conjunction with the AND operator. The expression preceding the AND is multiplied by the expression following it. Consider the program step

```
> 3.2 SET B = Z IF X AND Y = Z
```

If Z = 1, Y = 1, and X = -1, the test Y = Z is true (-1). The AND execution causes multiplication of -1 x 1, which tests true (-1). Therefore, B is set equal to Z, an undesired result. The step should be

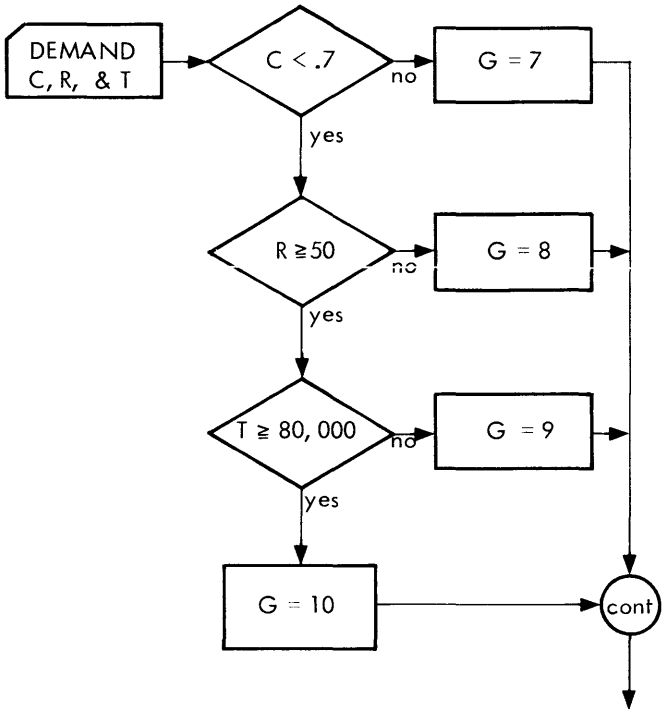
```
> 3.2 SET B = Z IF X = Z AND Y = Z
```

Example of a Steel Test Program

A certain steel is graded according to the results of three tests which are to determine whether it satisfies the following specifications:

Carbon content below .7
Rockwell hardness greater than 50
Tensile strength greater than 80,000 psi.

The steel is graded 10 if it passes all three tests, 9 if it passes only tests 1 and 2, 8 if it passes test 1 but fails test 2, and 7 if it fails test 1. The flow diagram is shown on the following page.




```
>1.0   DEMAND C, R, T (RET)
>1.05  SET G = 0 (RET)
>1.1   SET G = 10 IF T > = 80000 AND R > 50 (LF)
       AND C < .7 (RET)
>1.2   SET G = 9 IF NOT (T > = 80000) AND R > 50 (LF)
       AND C < .7 (RET)
>1.3   SET G = 8 IF NOT (R > 50) AND C < .7 (RET)
>1.4   SET G = 7 IF NOT (C< .7) (RET)
>1.44  LINE (RET)
>1.45  LINE (RET)
>1.5   TYPE IN FORM 1: G (RET)

>FORM 1: (LF)
  THE GRADE OF STEEL IS: %% (RET)
```

Then at execution:

```
>TO STEP 1.0 (RET)

     C = .6
     R = 55
     T = 85000

  THE GRADE OF STEEL IS: 10
>TO STEP 1.0 (RET)

     C = .6
     R = 55
     T = 75000

  THE GRADE OF STEEL IS: 9
>TO STEP 1.0 (RET)

     C = .6
     R = 45
     T = 85000

  THE GRADE OF STEEL IS: 8
>TO STEP 1.0 (RET)

     C = .8
     R = 55
     T = 85000

  THE GRADE OF STEEL IS: 7
```

Equivalent alternatives would be

```
(a)  1.0  DEMAND C, R, T
     1.1  SET G = 10 IF T > 80000 AND R > 50 AND C < .7
     1.2  SET G = 9 IF T < 80000 AND R > 50 AND C < .7
     1.3  SET G = 8 IF R < = 50 AND C < .7
     1.4  SET G = 7 IF C > = .7
     1.5  TYPE G

(b)  > 1.0  DEMAND C, R. T
     > 1.1  SET G = 10 IF C < .7 AND R > 50 AND
            T > = 80000
     > 1.2  SET G = 9 IF C < .7 AND R > 50 AND NOT
            T > = 80000
     > 1.3  SET G = 8 IF C < .7 AND NOT R > 50
     > 1.4  SET G = 7 IF C > = .7
     > 1.5  TYPE G
```

In the above example, the individual steps are tested as follows. The IF and AND operators as used in statement 1.1 above are written so that all conditions must test true. The computer checks by testing the left most conditional for true (1) or false (0), then proceeds to the right and tests the next conditional, etc. The result of each test (i.e., 0 or 1) is multiplied times the result of the next test, etc. If any part of the statement if false (0), it follows that the overall result is false (0). An example is given below.

```
T > = 80000   R > 50    C < .7
    85000       55        .6
True (1)    True (1)  True (1)  = (1) True = GRADE 10
```

To avoid confusion in determining how many conditions must be accounted for in writing logic programs, a truth table is constructed. A truth table shows a 1 if the condition is true, and a zero if the specified condition is false. A truth table for the grading of steel according to what combination of the three test conditions tests true or false is shown in Figure 10.

C < .7	R > 50	T > = 80000	GRADE
1	1	1	10
1	1	0	9
1	0	0	8
1	0	1	8
0	0	0	7
0	1	1	7
0	1	0	7
0	0	1	7

Figure 10. Truth Table for Steel Grading

THE "TO" COMMAND

In general, CAL executes the steps in a program in numerical order. TO is a command that alters the order in which CAL executes steps. TO provides a means for transferring control from one part of a program to another. When CAL obeys a TO command it goes to the part or step referred to by the TO, and executes successive steps until it encounters some other control change or the end of the program. The TO command permits the programmer to alter arbitrarily the sequence in which the program statements are executed.

The TO command may be thought of as a forward transfer or as a return transfer to cause repetition of certain steps in the program, as shown in the following example.

```
>1.0  DEMAND N (RET)
>1.1  SET I = 1 (RET)
>1.2  DEMAND X(I), Y(I) (RET)
>1.3  SET Z(I) = X(I) * Y(I) (RET)
>1.4  TYPE Z(I) (RET)
>1.5  SET I = I + 1 (RET)
>1.6  TO STEP 1.8 IF I > N (RET)
>1.7  TO STEP 1.2 (RET)
>1.8  PAUSE (RET)
```

Step 1.6 above is an example of a forward transfer. Forward transfers are almost always accompanied by a modifying conditional clause. Step 1.7 is an example of a backward transfer. Backward transfers are one way of getting repeat execution of a series of program steps.

MODIFYING CLAUSES OTHER THAN "IF" AND "FOR"

So far, commands such as DO and TO have been either unmodified or else were modified by either an IF or FOR clause. For example:

```
>1.0 DO PART 20 FOR I = 1 BY 2 TO 50
>2.0 TO STEP 23 IF J < = 5
```

Three other modifiers which add to ease of programming are also available in CAL. They are

UNLESS which allows the associated step to be executed if the logical evaluation of the modifying clause is zero.

WHILE which causes the associated step to be executed repeatedly if the logical evaluation of the modifying clause is zero.

UNTIL which causes the associated step to execute repeatedly if the logical evaluation of the modifying clause is zero.

USE OF THE "UNLESS" CLAUSE

The UNLESS clause is most often used for an iterative series of calculations where certain random exceptions exist. For example: An array called b contains mileages to certain cities from a point of origin and an array called $\bar{r}$ contains freight rates per lb per mile. It is desired to calculate the cost of moving 100 lbs to each of the cities from the point of origin and to place the results in a vector called $\bar{c}$, unless a mileage or rate does not exist (the program is to type a message to that effect). Such a program would appear as follows.

Assume the vectors $\bar{b}$ and $\bar{r}$ and the number of entries "N" in each are already loaded in the computer by part 1.0 which is omitted:

```
>2.0  DO PART 4.0 FOR I = 1 TO N (RET)
>2.1  TYPE C(I) FOR I = 1 TO N (RET)
>2.2  PAUSE (RET)
>4.0  TO STEP 4.4 UNLESS B(I) < = 0 (RET)
>4.1  TYPE "MILEAGE FOR THE FOLLOWING (LF)
      ENTRY IS ZERO OR NEGATIVE" (RET)
>4.2  TYPE I, B(I) (RET)
>4.3  SET C(I) = -88 (RET)
>4.35 DONE (RET)
>4.4  SET C(I) = B(I) * R(I) * 100 UNLESS R(I) < = 0 (RET)
>4.5  TO STEP 4.7 IF R(I) < = 0 (RET)
>4.6  DONE (RET)
>4.7  TYPE "RATE FOR THE FOLLOWING ENTRY IS (LF)
      ZERO OR NEGATIVE" (RET)
>4.8  TYPE I, R(I) (RET)
>4.9  SET C(I) = -99 (RET)
```

At execution, the program will type a message each time a mileage is missing, as a result of step 4.0, otherwise execution will transfer to step 4.4. Again, a message will be typed each time a rate is missing, as a result of the UNLESS clause testing true in step 4.4. Otherwise C(I) will be set to the correct cost to ship 100 lbs and the control subscript I will be incremented.

After the DO PART 4.0 has been evaluated for each value of I, the results typed out in step 2.1 are

(1) All cost successfully computed, and
(2) A code entry for all costs not successfully computed.

The values -88 or -99 are codes indicating the reason why the cost was not calculated (i.e., -88 means no mileage, -99 means no rate).

Note the way in which this program is written. If both mileage and rate are missing, only the fact that the mileage is missing will be recorded. In the next section, an alternate program having the additional feature to correct this flaw will be shown.

USE OF THE "WHILE" CLAUSE

By using the WHILE clause, a program may be written for recording the fact that both mileage and rate are missing (per the example above). The program (omitting the individual messages) would be as follows.

```
2.0  DO PART 4.0 FOR I = 1 TO N
2.1  TYPE C(I) FOR I = 1 TO N
2.2  PAUSE
```

```
4.0  SET C(I) = B(I) * R(I) * 100 WHILE B(I) > 0
     AND R(I) > 0
4.1  TO STEP 4.9 IF B(I) < = 0 AND R(I) < = 0
4.2  TO STEP 4.7 IF R(I) < = 0
4.3  TO STEP 4.5 IF B(I) < = 0
4.4  DONE
4.5  SET C(I) = -88
4.6  DONE
4.7  SET C(I) = -99
4.8  DONE
4.9  SET C(I) = -77
```

USE OF THE "UNTIL" CLAUSE

The most common use of the UNTIL expression is in conjunction with a FOR clause. For example, the step

```
1.0  DO PART 4.0 FOR I = 1  TO  N
```

can be written

```
1.0 DO PART 4.0 FOR I = 1 BY 1 UNTIL I > N
```

which is exactly equivalent.

A more useful utilization of UNTIL is in the case of a backward iteration through a loop. For example, if I were initially set equal to N and decreased in increments of 1 until I equaled 1, the step would be

```
1.0 DO PART 4.0 FOR I = N BY -1 UNTIL I = 1
```

The UNTIL clause is particularly useful when the length of a current vector is unknown or needs to be calculated. Consider the following example.

A manufacturer makes several assemblies which consist of two or more parts. The assembly listing is ordered from top to bottom, by assembly number, for indexing purposes. It contains the part number of each part required for each assembly, ordered from left to right. A segment of this list would be as shown in the two dimensional array, P, in Figure 11.

Assembly #	(Part # of Parts Making Up the Assembly)							
100	1	2	5	8	20	-1		
101	3	4	7	12	14	18	19	-1
102	6	2	-1					
103	1	6	7	-1				
104	13	16	18	21	-1			
105	10	11	12	-1				
106	8	4	3	12	22	23	-1	
⋮								
N								

Figure 11. Sample Assembly Parts Bill of Material List

Note that the horizontal dimension (J) in the array is different for each line entry and is terminated by a -1. A program to print a portion of the list from Assembly #100 to Assembly #120 (vertical dimension, (I)) would be as follows.

```
> 1.0  DO PART 4.0 FOR I = N BY 1 UNTIL (LF)
       P(I, 1) > 120 (RET)
> 1.1  PAUSE (RET)
> 4.0  DO STEP 4.2 FOR J = 1 BY 1 UNTIL (LF)
       P(I, J) < 1 (RET)
> 4.1  DONE (RET)
> 4.2  TYPE P(I, J) (RET)
```

MULTIPLE CLAUSES

Finally, IF, AND, FOR, WHILE, UNTIL, and UNLESS clauses can be combined in modifying a CAL command. Any combination can be used. For example, a program to calculate integral right triangles could be written as a single, direct CAL command.

```
>TYPE A, B, C IF FP(C) = 0 FOR C = SQRT(A↑2 + B↑2) (LF)
 FOR B = 1 TO A FOR A = 1 TO 12
```

In executing this rather massive command, CAL would take the following action: The right-hand (last) FOR variable, A, would be stepped through the values indicated. Then, for each value of A, B would be stepped through values up to A. For each value of B, C would, in turn, be SET to the value SQRT(A↑2 + B↑2). Finally, each time C was SET the IF clause would be tested, and if true, A, B, and C would be printed out. Note that the left-hand (first) FOR clause was used to SET a variable (C) to a single value rather than to generate a repeated operation.

Note that CAL reads the arguments of a command from left to right, but it reads the clauses that modify the command from right to left. In the example

```
> 3.7  DO PART 10 FOR A = 1 TO N IF C > D (LF)
 IF M < = 1 (RET)
```

CAL would first check to see if M is less than or equal to 1. Then, if that were true, it would next check to see if C is greater than D. If either clause were false, CAL would go on to the next step. Otherwise, part 10 would be executed for the indicated values of A.

CONDITIONALS AS EXPRESSIONS

In the CAL step below, the variable A on the left is set equal to the value of the expression on the right

```
1.0  SET A = SQRT(C↑2 - B↑2)
```

This type of replacement statement has often been used throughout the test examples so far. In this case, A is set to a specific value for each unique set of values assigned to B and C. Many times in programming it is desired to set A to a specific value that depends on a set of existing conditions. For example, suppose that if the following conditions of X exist, A is to be set to the value indicated.

X	Value to be SET in A
X < = 0	0
0 < X < = 10	1
10 < X < = 15	2
15 < X < = 30	3
30 < X < = 100	4
X > 100	5

Figure 12. Value of A as a Function of X

One way of writing this program would be

```
> 1.0  SET A = 0 IF X < = 0 (RET)
> 1.1  SET A = 1 IF X > 0 AND X < = 10 (RET)
> 1.2  SET A = 2 IF X > 10 AND X < = 15 (RET)
> 1.3  SET A = 3 IF X > 15 AND X < = 30 (RET)
> 1.4  SET A = 4 IF X > 30 AND X < = 100 (RET)
> 1.5  SET A = 5 IF X > 100 (RET)
> 1.6  TYPE A, X (RET)
```

CAL has a powerful feature that makes conditional expressions possible in the example above. Using a conditional expression, the foregoing can be achieved in a single step.

```
1.0 SET A=IF X<=0 THEN 0 ELSE IF X>0
AND X <= 10 THEN 1 ELSE IF X>10 and X <=15
THEN 2 ELSE IF X>15 AND X<=30 THEN 3 ELSE
IF X>30 AND X <=100 THEN 4 ELSE 5
```

Conditional expressions take the general form

SET v=IF e_{p1} THEN e_{v1} ELSE IF e_{p2}
THEN e_{v2}... ELSE e_{vn}

When a conditional expression is evaluated, e_{p1} is first evaluated. If it is non-zero, e_{v1} is evaluated and its value becomes the value of the expression. If e_{p1} is zero, e_{v1} is ignored and e_{p2} is evaluated. If the expression ends with ELSE e_{vn}, then the value of e_{vn} will be the value of the expression if all the e_{pi} are 0. The expression may, however, end with

IF e_{pn} THEN e_{vn}

In this event, the value of the expression is 0 if all the e_{pi} equal 0. Note that the only expressions actually evaluated are those whose values are required in determining the value of the expression. As another example, consider the statement

```
>SET B=IF X MOD 2=0 THEN-1 ELSE IF X > 10 (LF)
 THEN 0 ELSE 1 (RET)
```

This statement will set B to -1 if X is an even integer, to 0 if X is not an even integer and is greater than 10, and to 1 if X is not an even integer and is less than 10.

THE "WHERE" MODIFIER

A WHERE modifier can be appended to an expression in order to set the value of a variable to the value of a specified expression. The general form is

SET V = **e** WHERE $v_1 = e_1$

The modifier, which is evaluated first, sets the value of v to the value of $v_1 = e_1$. For example, the step

```
1.0 SET A = SIN(Z)/COS(Z) WHERE Z = ATAN(X, Y)
```

has the value

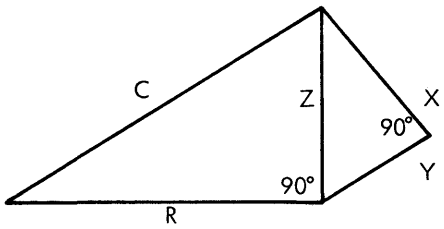
SIN(ATAN(X, Y))/COS(ATAN(X, Y))

In the above example, Z will be continuously evaluated. However, it is possible to have the WHERE modifier evaluated only once, by placing a comma before the WHERE. This is particularly useful in the iteration. For example:

```
>SET X = N(I)/Q FOR I = 1 TO 10, WHERE (LF)
 Q = SQRT(64) (RET)
```

The WHERE modifier may be followed by more than one expression; each must be separated by an ampersand sign (&) (See the last example in this section for an example of its use).

In a previous example, a program was written to find C according to the following figure.




```
> 1.0  DEMAND X, Y, R (RET)
> 2.0  SET C = SQRT(R↑2+Z↑2) WHERE (LF)
  Z = SQRT (X↑2+Y↑2) (RET)
>3.0  TYPE X, Y, R, Z, C (RET)
```

As an additional example of the use of WHERE, consider a program to evaluate

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{N!}$$

Given X and the number of terms, N, the program without a WHERE clause could be written as

```
> 1.1  DEMAND X, N (RET)
> 1.2  SET T = 1 (RET)
> 1.3  SET S = 1 (RET)
> 1.4  DO PART 4 FOR I = 1 UNTIL T < 10E-8 (RET)
> 4.2  SET T = T*X/I (RET)
> 4.3  SET S = S+T (RET)
> 5.43 TYPE S, T, X, N (RET)
> 5.7  TO STEP 1.1 (RET)
```

A program to compute e^x with accuracy better than 10^{-8} utilizing WHERE clauses can be written in one step as

```
>1.0 SET E=1+SUM(I=1 BY 1 UNTIL T<10E-8: (LF)
 T WHERE T=X↑I/F WHERE F=I*F) (RET)
```

where T=1 and F=1.

4. SPECIAL STATEMENTS (INCLUDING FILE CONSTRUCTION) AND SUBPROGRAMS

SPECIAL STATEMENTS

The discussion thus far has covered statements which may be either direct or indirect.† Remember that a direct statement is one that does not have a step number and therefore results in a one-time execution (i.e., the statement is not saved in a program). An indirect statement does have a step number and is saved in a program. Execution of an indirect statement does not occur until the program is executed. The most common two-way (direct and indirect) statements use the TYPE and SET commands. Used as direct statements, for example, these commands would appear as

```
> SET C = 4 (RET)
> SET B = 3 (RET)
> TYPE A = SQRT(B↑2+C↑2) (RET)
```

The computer would immediately execute the commands and print

```
A = SQRT(B↑2+C↑2) = 5
```

In program form, the same commands would appear in indirect statements as

```
> 1.1 SET C = 4 (RET)
> 1.2 SET B = 3 (RET)
> 1.3 TYPE A = SQRT(B↑2+C↑2) (RET)
```

Then, to initiate a single execution of the program, the user would type a direct statement such as TO STEP 1.1.

In addition to statements which may be either direct or indirect, CAL has certain statements which may be either direct only or indirect only. The purpose of this section is to define and illustrate the use of these special statements.

SPECIAL DIRECT STATEMENTS

The following commands may be used in direct statements only.

DELETE STEP i.j where i.j is any statement number.

This command causes the specified statement to be deleted.

DELETE PART i where i is any expression or part number.

This command causes the specified part (with all its steps) to be deleted. In the case of expressions, it causes the expression to be evaluated and the part equal to the integer portion to be deleted (this latter point will become clear in the examples to follow).

DELETE FORM i where i is any form number or expression

This command causes the specified form number to be deleted; in the case of expressions, it causes the expression to be evaluated and the form equal to the integer portion to be deleted.

DELETE i where i is any variable

This command causes the value of the indicated variable to be deleted without changing the program.

DELETE ALL

This command causes the entire program in process to be destroyed and returns the user to the CAL beginning sequence.

As an example of the DELETE statement, consider the following program segment.

```
>1.70 DO PART 10 FOR I = 1 TO N (RET)
>1.75 TO STEP 1.0 (RET)
>1.80 TYPE "EQUATIONS ARE SINGULAR" (RET)
>1.85 TO STEP 1.0 (RET)

>2.00 TYPE IN FORM 2:I (RET)
>2.10 DEMAND A(I, J) FOR J=1 TO N (RET)

>3.00 SET L=0 (RET)
>3.10 SET G=0 (RET)
>3.20 DO PART 4.0 FOR I=J TO N (RET)
>3.3 SET R(G)=R(J) (RET)
>3.4 SET R(J)=G (RET)
>3.5 DO PART 5 FOR K=J+1 TO N (RET)

>4.0 TO SETP 4.2 IF ABS(A(R(I), J)) > L (RET)
>4.1 TO STEP 4.4 (RET)
>4.2 SET L=A(R(I), J) (RET)
>4.3 SET G=R(I) (RET)
>4.35 TYPE L, G, R, A, J (RET)
>4.4 SET I=I (RET)

>5.0 TO STEP 5.4 IF ABS(A(R(K), J)) < 1E-5 (RET)
>5.1 SET M=A(R(K), )/A(R(J), J) (RET)
>5.2 DO PART 6.0 FOR L=J TO N (RET)
>5.3 SET B(R(K))=B(R(K))=M*B(R(J)) (RET)
>5.4 SET K=K (RET)

>6.0 SET A(R(K), L)=A(R(K), L)=M*A(R(J), L) (RET)
>6.1 SET A(R(K), L)=0 IF ABS(A(R(K), L))<. 000001 (RET)

>7.0 TO STEP 1.8 IF ABS(A(R(I), I)) < 1E-5 (RET)
>7.1 DONE (RET)
```

†In examples given in previous chapters, statements were used that did not meet this criteria. These statements, used to complete a program or program segment, will be fully discussed in this chapter.

The following DELETE commands are now executed.

```
DELETE STEP 1.75 (RET)
DELETE STEP 1.85 (RET)
DELETE STEP 2 (RET)
DELETE STEP 4.4 (RET)
```

A relisting of the program would now appear as

```
1.70  DO PART 10 FOR I=1 TO N
1.80  TYPE "EQUATIONS ARE SINGULAR"

2.10  DEMAND A(I, J) FOR J=1 TO N

3.00  SET L=0
3.10  SET G=0
3.20  DO PART 4.0 FOR I=J TO N
3.3   SET R(G)=R(J)
3.4   SET R(J)=G
3.5   DO PART 5 FOR K=J+1 TO N

4.0   TO STEP 4.2 IF ABS(A(R(I), J)) > L
4.1   TO STEP 4.4
4.2   SET L=A(R(I), J)
4.3   SET G=R(I)
4.35  TYPE L, G, R, A, J

5.0   TO STEP 5.4 IF ABS(A(R(K), J)) < 1E-5
5.1   SET M=A(R(K), J)/A(R(J), J)
5.2   DO PART 6.0 FOR L=J TO N
5.3   SET B(R(K))=B(R(K))=M*B(R(J))
5.4   SET K=K

6.0   SET A(R(K), L)=A(R(K), L)=M*A(R(J), L)
6.1   SET A(R(K), L)=0 IF ABS(A(R(K), L)) < .000001

7.0   TO STEP 1.8 IF ABS(A(R(I), I)) < 1E-5
7.1   DONE
```

The DELETE PART command is now executed as shown.

```
DELETE PART 4.0 (RET)
DELETE PART 7.0 (RET)
```

The program would now appear as

```
1.70  DO PART 10 for I=1 TO N
1.80  TYPE "EQUATIONS ARE SINGULAR"

2.10  DEMAND A(I, J) FOR J-1 TO N

3.00  SET L=0
3.10  SET G=0
3.20  DO PART 4.0 FOR I=J TO N
3.3   SET R(G)=R(J)
3.4   SET R(J)=G
3.5   DO PART 5 FOR K=J+1 TO N

5.0   TO STEP 5.4 IF ABS(A(R(K), J)) < 1E-5
5.1   SET M=A(R(K), J)/A(R(J), J)
5.2   DO PART 6.0 FOR L=J TO N
5.3   SET B(R(K))=B(R(K))=M*B(R(J))
5.4   SET K=K

6.0   SET A(R(K), L)=A(R(K), L)=M*A(R(J), L)
6.1   SET A(R(K), L)=0 IF ABS (A(R(K), L)) < .000001
```

If the DELETE PART command had specified an expression such as A*B+2.5, where A=1 and B=2, the command would be

```
DELETE PART A*B+2.5
```

This would be interpreted as

```
(a)  DELETE PART 1x2+2.5
(b)  DELETE PART 4.5
```

which would be equivalent to the integer portion of 4.5, or

```
DELETE PART 4.0
```

If the command DELETE ALL is executed, the whole program segment is completely destroyed so that the storage is completely free for reuse. At execution of the DELETE ALL command, the computer prints

```
CAL
NUMBER OF STATEMENTS NEEDED =
HEADING, PLEASE
```

Note that this is the same result as when CAL is initiated after LOG IN.

The DELETE ALL command is especially useful to correct a type of error common to inexperienced programmers. For example, a program to calculate the square roots of a series of numbers is written and appears as

```
>1.0  DEMAND N (RET)
>1.2  DO PART 2 FOR I=1 TO N (RET)
>1.3  TO STEP 3.0 (RET)

>2.0  DEMAND A(I), B(I) (RET)
>2.2  SET C(I)=SQRT(B(I)↑2+A(I)↑2) (RET)
>2.4  TYPE B(I), A(I), C(I) (RET)

>3.0  DONE (RET)
```

This program is executed successfully. Now without executing the DELETE ALL command, the programmer proceeds to write another program to find the average of a series of numbers.

```
>1.0  DEMAND N (RET)
>1.1  DO PART 2 FOR I=1 TO N (RET)
>1.3  SET F=0 (RET)
>1.4  SET V=F/M (RET)

>2.0  DEMAND A(I), B(I) (RET)
>2.1  SET C(I)=A(I)*B(I) (RET)
>2.3  SET F=F+C(I) (RET)
>2.4  TYPE B(I), A(I), C(I) (RET)
```

At execution, the program would appear to CAL as

```
1.0  DEMAND N
1.1  DO PART 2 FOR I=1 TO N
1.2  DO PART 2 FOR I=1 TO N
1.3  SET F=0
1.4  SET V=F/M
```

```
2.0 DEMAND A(I),B(I)
2.1 SET C(I)=A(I)*B(I)
2.2 SET C(I)=SQRT(B(I)↑2+A(I)↑2)
2.3 SET F=F+C(I)
2.4 TYPE B(I),A(I),C(I)
```

Notice that every place in which the second program had a number exactly the same as the first program, that step was replaced. However, wherever the second program had a gap in its numbering (i.e., 2.1, —, 2.3) and this number gap was equivalent to a number in the first program, the first program remained intact. To clarify, 2.0 of the second program replaced 2.0 of the first (the fact that they were the same statement is coincidental). 2.1 is from the second program, whereas 2.2 is from program one. The end result is that program two finds the average of

```
C(I)=SQRT(A(I)↑2+B(I)↑2) FOR I=1 TO N,
```

but the programmer's intent was to find the average of

```
C(I)=A(I)*B(I) FOR J=1 TO N
```

This overlay feature has several editing advantages which will be discussed later. In this case, however, if program one is to be saved, the procedure should be

(1) Load program one to the disc file.

(2) DELETE ALL (this is necessary as the program is still in core).

(3) Then proceed to write program two.

Other direct statements include those which utilize the GO and STEP commands.

GO

This command permits continuation of execution after an interrupt, an executor's error, or after a PAUSE statement is executed (the PAUSE statement is explained below in the section on indirect statements, which also includes an example of how the GO command works).

STEP

This command causes the next statement of the program to be executed, after which control returns to CAL (as though a PAUSE had been executed). For example, the user has typed the following sequence of statements.

```
>1.0 PAUSE (RET)
>1.1 SET A=SIN(A+SIN A) (RET)
>1.2 SET B=SIN(A+SIN B) (RET)
>1.3 SET D=SIN(B+SIN D)(RET)
```

The following would then be useful for debugging these statements by executing them step by step and observing the variables. A, B and D must have values.

```
> TO STEP 1.0 (RET)
PAUSE IN STATEMENT 1.0
> STEP (RET) (Step 1.1 would be executed)
> TYPE A (RET)
A=9.7
> STEP (RET) (Step 1.2 is executed)
> TYPE B (RET) (request value of B)
B=0.9
```

Thus, by direct interaction, the user may correct his program a step at a time if he so desires.

SPECIAL INDIRECT STATEMENTS

The commands discussed in this section (PAUSE and DONE) may be used in indirect statements only; i.e., those which require a statement number and are stored in program form.

PAUSE

This command causes a message to be printed out and program execution to stop. During the pause, the user may make corrections, etc. The command may be made conditional by utilizing modifiers. As an example of the use of PAUSE, consider the following program to read in a two dimensional array of numbers whose dimensions are N and M.

```
1.1 DO PART 2 FOR I=1 TO N
1.1 TYPE "AT PAUSE MAKE CORRECTIONS AND TYPE 'GO'"
1.2 PAUSE
1.3 TO STEP 3.0
```

If N=2 and M=3, the following would occur at execution.

```
A(1, 1)=2
A(1, 2)=4
A(1, 3)=3
A(2, 1)=5
A(2, 2)=6
A(2, 3)=2
AT PAUSE MAKE CORRECTIONS AND TYPE 'GO'
PAUSE AT STEP 1.2
>
```

Suppose that the values of A(2, 1) and A(2, 2) were supposed to be 6 and 5, respectively, instead of 5 and 6. The > sign directly below the PAUSE statement indicates that CAL will accept a direct command. Thus, at this point the user may type

```
> SET A(2,1)= 6 (RET)
> SET A(2, 2)=5 (RET)
```

By typing GO at this point, execution will begin again at step 1.3, which indicates that the program should transfer to step 3.0. The corrected values have now replaced the original entries.

The PAUSE statement is also useful in terminating a program. For example, the last two statements in a program might be

```
>20.0 TYPE "IF RE-EXECUTION IS DESIRED TYPE (LF)
      'TO STEP1.0' OTHERWISE HIT THE ESCAPE (LF)
      KEY ONCE TO RETURN TO CAL, TWICE TO (LF)
      RETURN TO THE EXECUTIVE MONITOR" (RET)

>20.1 PAUSE (RET)
```

At the execution of steps 20.0 and 20.1 the message would be printed, plus

```
PAUSE AT STEP 20.1
>
```

The program will re-execute if the user then types

> TO STEP 1.0 (RET)

If the user wished to terminate the program he would follow the instructions given in the message.

DONE

This statement is ignored unless a DO PART statement is in force, in which case the DO PART is terminated. Since DO PARTS are used many times in iterative loops, the DONE command is frequently used to terminate a loop. For example, an array ($\bar{q}$) contains quantities of parts shipped to customers and a corresponding array ($\bar{p}$) contains the unit price (customer identity has a one to one correspondence with the subscript value; i.e., when the subscript is "i" this corresponds to a code "i" for that customer's name). If a zero value occurs in either the $\bar{q}$ array or the $\bar{p}$ array, the calculation is to move to the next set of values. The program segment to perform this, assuming the arrays $\bar{q}$ and $\bar{p}$ have already been input to the program, would be

```
> 3.0 DO PART 4 FOR I = 1 TO N[t] (RET)
> 4.0 DONE IF Q(I) = 0 OR P(I) = 0 (RET)
> 4.1 SET C(I) = A(I)*P(I) (RET)
> 4.2 TYPE I, C(I) (RET)
```

At execution, whenever Q(I) or P(I) are zero, the DONE command will return execution to the DO command in step 3.0. The DO command causes an incrementation of I and then re-execution of part 4.0.

SUBPROGRAMS

Mathematical functions such as SQRT, EXP, etc., are predefined functions and, as such, are always available in CAL. However, there are often functions or special procedures that are not predefined but would be useful for a program being written. It is then up to the user to define his own functions. This is possible in CAL and can be done in several ways. The most common way is to construct and use CAL "subprograms". Each subprogram, when supplied to the computer along with the main program, will then serve as if it were a predefined function.

Subprograms are somewhat independent and can thus be tested separately in many cases. Groups of tested subprograms forming progressively larger subsystems can be tested separately until finally the entire system of programs may be proven as a working unit. This independence of parts means that several programmers may work effectively under one supervisor on separate subsections of a large project, depending on the nature of the project.

Another useful property of subprograms is that they may always be added to the "library" of a computer system when they prove useful enough. Thus they become predefined functions for all programs written in the future.

[t] N equals total entries in the array.

Statements of the type previously discussed can be combined to form a subprogram. Consider the following program which calculates the product of N*(N-1)*(N-2)...1. The product is called the factorial of N.

```
> 1.0 SET N = 10 (RET)
> 2.0 DO PART 5.0 (RET)
> 2.1 TYPE P (RET)
  .
  .
  .
> 5.0 SET P = PROD (I=1 TO N:I) (RET)
```

In this example, Step 5.0 is called the subprogram, Step 2.0 is the call to the subprogram, N is the argument, and P is the result of the subprogram.

This type of subprogram has a definite limitation, in that only the factorial of N can be calculated by step 5.0. It would be more convenient to have a subprogram that could calculate the factorial of any variable. It is possible to construct such subprograms in CAL by using DEFINE statements (which are always written without step numbers). In this way, CAL lets the user define a step or many steps in a general sense so that any variable or variables can be acted upon.

ONE-STEP SUBPROGRAMS

To define the subprogram that consists of one step only, the user uses the following form of the DEFINE statement.

DEFINE f $[v_1, v_2, \ldots, v_n]$ = e

where f is any variable name, v is any variable and e is any expression. For example

DEFINE F [A, B, C] = A*B+C

The call to a subprogram is the appearance of the function name in a statement, which initiates the execution of the subprogram. Either of the following statements, for example, would call the subprogram for execution:

1.0 SET X = Y + Z + F [1, 2, 3]

or

TYPE F[X, Y, Z+5]

The arguments of the subprogram are included in brackets in the call. They may be variables, constants, or expressions. The variables in brackets in the original definition are called local variables or dummy arguments. There must always be the same number of arguments as local variables.

Note that while the DEFINE statement is always a direct statement, the call to the subprogram may appear in either a direct or indirect statement.

Consider again the factorial example, N*(N-1)*(N-2)...1. Using the DEFINE statement, the subprogram could be written as

DEFINE F [N] = PROD(I FOR I = 1 TO N)

To execute the subprogram for N = 10, the user could type

```
TYPE F [10]
```

We can consider another example.

```
DEFINE F [X, Y, Z, W] = X↑W + Y/Z
```

The call

```
TYPE F [3, 4, 2, 2]
```

would cause the computer to evaluate X↑W + Y/Z, using the values 3, 4, 2, and 2 for X, Y, Z, and W, respectively, and to print the result. Thus F = 3↑2 + 4/2 = 11, and 11 would be printed as the result of subprogram F.

Using the same subprogram definition, the following steps could also be used to cause the computer to print the same result.

```
> SET A = 3 (RET)
> SET B = 4 (RET)
> SET C = 2 (RET)
> SET D = 2 (RET)
> TYPE F [A, B, C, D] (RET)
```

The effect of this call is that the value of A has been assigned to X, B to Y, C to Z, and D to W. In this case, the arguments are A, B, C, and D, while the local variables are X, Y, Z, and W.

The local variables in a subprogram are not related to any variables in the main program, even if they have the same name. Local variables are assigned no location in memory. Their function is only to serve as placeholders. This point can be made clear with another example.

```
>DEFINE E [I, J, K, L, M] = (I+J+K)/(L+M) (RET)
>1.0 SET I = 10 (RET)
 .
 .
 .
>2.0 TYPE E [1, 2, 3, 4, 5] (RET)
 .
 .
 .
>5.0 TYPE I (RET)
```

Note that I appears as a variable in the main program and as a dummy variable in the definition statement. The value of I in the main program is set to 10 before the subprogram is executed. Thus, the main variable I will have the value of 10 when step 5.0 is executed, even though the dummy variable I is given the value of 1 in step 2.0. When the subprogram is executed by the call in step 2.0, I will have the value of 1 but this will not affect the I of step 5.0. This characteristic is true only in the case of local variables.

MULTIPLE-STEP SUBPROGRAMS

To write subprograms longer than one step, another form of the DEFINE statement is available.

```
DEFINE f [v1, v2, ..., vn]: statement
```

where f is any variable name and v is any variable. For example

```
DEFINE X [A, B, C] : TO PART 5
```

or

```
DEFINE W [M, N] : TO STEP 4.2
```

Again, the DEFINE statement is always a direct one. When used in this form, the call must always be an indirect statement. The following is an example of a valid call to a multi-step subprogram.

```
1.0 SET F = W [M, N]
```

The local variables have the values of the arguments, as described above. However, to terminate the subprogram the user must include a RETURN statement, which will also assign the value of e to the variable name, f. The form of the statement is

```
RETURN e
```

where e is any expression.

As an example of subprograms with more than one step, consider the sales tax program in chapter 3. This would appear as a subprogram as follows.

```
>DEFINE S [C]: TO STEP 10.0 (RET)
>1.0 DEMAND X (RET)
>2.0 TYPE S [X] (RET)
 .
 .
 .
>10.0 SET T = .04*IP(C) (RET)
>10.1 SET T = T+.01 IF FP(C) > .15 (RET)
>10.2 SET T = T+.01 IF FP(C) > .37 (RET)
>10.3 SET T = T+.01 IF FP(C) > .62 (RET)
>10.4 SET T = T+.01 IF FP(C) > .84 (RET)
>10.5 RETURN T (RET)
```

All of part 10 is executed with the value of the argument X assigned to local variable C. Step 10.5 assigns the value T to S and returns control to step 2.0.

In terminating a subprogram with the RETURN statement, the user must take care that the range of any DO or FOR commands are complete before the RETURN appears. Otherwise,

the part will not be completed. The following example illustrates the correct placement of a RETURN statement.

```
>DEFINE Z [N]: TO STEP 10.0 ↵
>1.0  SET X = Z [30] ↵
  .
  .
  .
>10.0 SET S = 0 ↵
>11.0 SET Q = 0 ↵
>12.0 DO PART 14 FOR I = 1 TO N ↵
>13.0 TO STEP 15.0 ↵
>14.0 DEMAND A(I) ↵
>14.1 SET S = S + A(I) ↵
>14.2 SET Q = Q + A(I)*A(I) ↵
>15.0 TYPE S/N ↵
>16.0 TYPE SQRT (Q - ((S*S)/N)/N-1) ↵
>17.0 RETURN S ↵
```

This subprogram reads N numbers, and calculates and prints the mean and standard deviation. Execution begins with step 10.0 and continues to step 12.0 which sends it to part 14. Part 14 is completed before execution is returned to step 13.0. Execution ends with steps 15, 16, and 17. If the RETURN had been included in part 14, within the range of the DO command, CAL would reject the program for syntax error.

A RETURN is never used when the subprogram is defined with a DO statement. The reason for this is that DO PART or DO STEP is not finished until the whole subprogram has been completely executed. Thus, the RETURN statement is unnecessary. Consider the following example:

```
>DEFINE S [C]: DO PART 10.0 ↵
>1.0  DEMAND X ↵
>2.0  SET P = S [X] ↵
>3.0  TYPE T ↵
  .
  .
  .
>10.0 SET T = .04*IP(C) ↵
>10.1 SET T = T + .01 IF FP(C) > .15 ↵
>10.2 SET T = T + .01 IF FP(C) > .37 ↵
>10.3 SET T = T + .01 IF FP(C) > .62 ↵
>10.4 SET T = T + .01 IF FP(C) > .84 ↵
```

Note that in this example, the purpose of step 2.0 is to initiate the program. P and S, after execution, have the value of zero. Therefore, step 3.0 was necessary to print the "result" of the subprogram.

This section has presented the basic rules for constructing subprograms. Increasing familiarlity with the DEFINE statement will suggest other useful possibilities for subprograms. For example, the following subprogram definition allows the user to choose, at execution time, which sections of statements to execute.

```
DEFINE F [I]: TO STEP I
```

PROGRAM FILES

Programs may be saved in the form of files. Each user may have a number of his own files which are private to him. The command to save a program written in CAL is

```
>DUMP ↵
TO/file/ ↵
  NEW (old) FILE ↵
```

This command takes all the steps and forms in memory and writes them onto the specified file. To bring the steps back from the file into CAL, the user types

```
> LOAD ↵
FROM /file/ ↵
```

For more information about program files, see the Terminal User's Guide.

5. INPUT/OUTPUT — DATA FILE CONSTRUCTION

One of the most powerful features of CAL is that it allows the user to input numerical information while the program is in the execution mode. CAL also provides instructions for formatting both teletype input and output information and, finally, has the capability of reading and writing data files.

THE "DEMAND" STATEMENT

The DEMAND statement is the executable command which allows numerical input by the user during program execution. The general form of the expression is

DEMAND (variable), (variable), . . . , (variable) (RET)

The command causes each variable name to be typed out one at a time. As each variable on the list comes up, CAL waits for the user to input a value for that variable. The variable request list generated by CAL is in the order that the variables appeared in the DEMAND statement.

>DEMAND X, Y, Z, A, R (RET)

At execution, CAL causes the computer to print

X = (user enters numeric value) (RET)
Y = (user enters numeric value) (RET)
Z = (user enters numeric value) (RET)
A = (user enters numeric value) (RET)
R = (user enters numeric value) (RET)

When numerical data is being typed in response to the DEMAND statement, any non-numeric characters typed before the number will be ignored. For example

>DEMAND X, Y (RET)
X = A3
Y = LM4

would assign 3 to X, and 4 to Y.

If a mistake is made during the typing of a number, the entry may be deleted by striking Q^c, which deletes all characters typed so far and allows the number to be retyped. To terminate a unique entry, the character following the number must be a carriage return, space, comma, or semicolon. If a carriage return is used as a terminator, the next data request will occur on the next line. When a space is typed, the next data request will appear on the same line if additional data requests are yet to be satisfied in the particular demand statement being executed.

"DEMAND" STATEMENTS WITH MODIFIERS

The DEMAND statement may be used in combination with modifier clauses, such as FOR, IF, WHERE, UNTIL and WHILE.

For example, the DEMAND statement used with a FOR clause would look and execute as follows.

4.1 DEMAND A(I) FOR I = 1 TO 4 (RET)

At execution the computer will print

A(1) = (Operator enters number plus (RET))
A(2) = (Operator enters number plus (RET))
A(3) = (Operator enters number plus (RET))
A(4) = (Operator enters number plus (RET))

A two dimensional matrix, $\bar{A}$, with dimensions of m rows and n columns may be input by a DEMAND statement as follows.

1.0 DO PART 2.0 FOR I = 1 TO M (RET)
2.0 DEMAND A(I, J) FOR J = I TO N (RET)

At execution, the computer will start with "I" set equal to one and will request the elements in the first row for J = 1 TO J = N, then "I" will be set to the value 2 and the sequence in J repeated, e. g.,

A(1, 1) =
⋮
A(1, N) =

A(2, 1) =
⋮
A(2, N) =

A(M, 1) =
⋮
A(M, N) =

Another example would be

5.2 DEMAND X(I) FOR I = 1 TO 100 WHILE (LF)
X(I-1)#0, WHERE X(0) = 1

X(1) =
X(2) =
X(3) =
⋮
(this will continue until I = 100 or the input is zero)

THE "FORM" STATEMENT FOR INPUT

CAL allows the user to control the form of his input by using the command

DEMAND IN FORM e: $v_1, v_2, \ldots, v_n$ (where e is any expression and v any variable)

and its corresponding

FORM n: (LF)
(where n is a constant, the numerical evaluation of the expression e)

Often programs are written by one person and used by many. When this is the case, the type of input required must be carefully explained to the user, so he knows when and what to type. This can be accomplished by using the FORM and DEMAND IN FORM commands.

The form and demand in form commands work by evaluating the expression e. The corresponding form statement defines what will be printed and how many numbers will be accepted. All text (including blanks) in the form statement is printed. One number (in either decimal or scientific notation) is accepted for each "#" sign. The "#" sign is not printed.

As an example, three variables to be input represent a unit cost (C), a quantity (Q), and a discount (D). The CAL statements are

```
1.1    DEMAND IN FORM 1: C,Q,D(RET)
FORM 1: (LF)
UNIT COST = #  QUANTITY = #  DISCOUNT = #
```

At execution, the computer would print the first portion of the text and then wait for the user to type the first number. Then the computer prints the second portion of text and waits for the second number to be typed, and so on.

```
UNIT COST = 9.65  QUANTITY = 500  DISCOUNT = .05
```

The user terminates each number with a comma, semicolon, or blank. (Note that the user types a line feed (not a carriage return) after FORM 1:).

The same example could also appear as

```
>1.1 DEMAND IN FORM 1: C,Q,D (RET)
>FORM 1: (LF)
 UNIT COST = # (LF)
 QUANTITY = # (LF)
 DISCOUNT = # (LF)
```

Similarly, at execution the computer would type the first portion of text and wait for a number.

```
UNIT COST = 9.65(RET)
QUANTITY = 500 (RET)
DISCOUNT = .05 (RET)
```

However, in this example, when a space, comma, or semicolon following the number is typed by the user, the computer will print a carriage return, line feed and the next portion of text.

Note that the FORM statement never has a line number.

One "#" sign should appear in the FORM statement for each variable in the demand list.

The FORM statement is also used by those who want no printing at all before input. This is particularly useful when the data to be typed is on a paper tape. Consider the following example.

The $\overline{A}$ matrix used in one of the previous examples could be read, one row at a time, across the paper by using the following.

```
>1.0 DO PART 2.0 FOR I = 1 TO M(RET)
>2.0 DEMAND IN FORM 1: A(I,J) FOR J = 1 TO N(RET)

 FORM 1: (LF)
 #   #   #   #   #   #   #   #   #
```

In this case, the variable name is not printed. The computer will expect a number (in any form) for each #. The user might type

```
9.1, 8.3, 7.2,.....6.0 (RET)
3.2,.....4.2 (RET)

3.2,.....3.1 (RET)
```

As before, each number may be terminated by a space, comma, or semicolon.

DATA INPUT FROM PAPER TAPE

This data could have been typed directly or typed off line onto a paper tape. The latter method is often used to reduce typing time. The procedure to follow for this example is: (1) prepare an off-line tape by punching the value of $a_{1,1}$ (comma, semicolon or blank) $a_{1,2}$ (comma, semicolon or blank) to $a_{1,n}$ (carriage return, etc., and (2) press the tape reader button, turn the TD switch on and the program will accept the tape at full teletype speed. The computer is unable to distinguish between input from paper tape and direct typing.

THE "FORM" STATEMENT FOR OUTPUT

The TYPE statement may also be used with the FORM statement.

```
TYPE IN FORM 1:
   :
   .
FORM 1: (LF)
    %%%%.%   %%%%.%%
```

Note again that the user types a line feed and the computer supplies the carriage return.

The significance of the percent sign is to tell the computer in what form the number should be printed.

For example

```
%%%.%     tells the computer to print a sign, two
     digits, a decimal point, and then one digit.
-42.3
(+)21.9   (plus signs are not printed)
49.0
-1.5      (no leading zeros are printed)
```

Text to be printed may also appear in a FORM statement.

For example

```
>FORM 2 (LF)
  VOL = %%%%.%     AREA = %%%%.% (RET)
```

When the following command is executed (where V = 452.1 and A = 205.9)

```
>TYPE IN FORM 2 : V,A (RET)
```

the computer will type

VOL = 452.1 AREA = 205.9

The percent sign will always produce a number in decimal notation. To produce a number in scientific notation, the "#" sign is used.

The syntax of the FORM statement for numbers in scientific notation is

```
>FORM 1: (LF)
  ######, ######, ..., ###### (RET)
```

A minimum of SIX "#" symbols is required to denote a scientific notation field. The six "#" symbols are required to provide space for the sign, the decimal point, at least one significant digit, the "E" denoting a power of ten, the sign of the exponent, and the exponent itself. For example, to output the number 1.1E-3, 30.13+12, and 50.265E-5, the form statement would be

```
>FORM 1: (LF)
  ######, ########, ######### (RET)
```

Note that the fields in either decimal or scientific notation used in the FORM statements, are delimited by a blank. If commas appear, as in the example above, they will be printed.

As described, a FORM statement is uniquely identified by a number appearing directly after the word "FORM". The syntax rules for FORM statements for output are

- The identification number must be immediately preceded by a space and followed by a colon.
- The colon is always followed by a line feed (LF) carriage return (RET) and then the field designation.
- Two types of field designators are available: decimal number (the % symbol) and scientific notation (the # symbol).
- A form statement may have several field lines; each intermediate line must be terminated by line feed-carriage return, and the last line by a carriage return-line feed.
- Text may appear in the numerical fields.

Another example is given below.

```
>29 TYPE IN FORM 1: C,Q,D (RET)
>FORM 1: (LF)
  UNIT COST = %%.%%,   QUANTITY=%%%%., (LF)
  DISCOUNT =%.%% (RET)
```

At execution the format would appear as

UNIT COST = 1.25 QUANTITY = 235
DISCOUNT = .05

If column form instead of row were desired the FORM statement would have to be written as

```
>FORM 1: (LF)
  UNIT COST = %%.%% (LF)
  QUANTITY = %%%. (LF)
  DISCOUNT = %.%% (RET)
```

If there are more numbers in the TYPE statement than fields in the form, the form is reused as often as necessary. If a FOR modifier is used immediately after the list of expressions, the form will not be initialized each time around for FOR. Instead, output will take place as though all the expressions generated by the FOR had been written in the TYPE statement. Thus

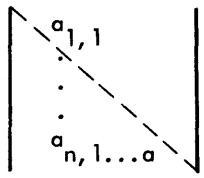
```
>FORM 1: (LF)
  %%%% %%%% %%%% %%%% %%%% %%%% (RET)

>TYPE IN FORM 1: I↑2 FOR I = 1 TO 14 (RET)
```

will result in

```
  1     4     9    16    25    36
 49    64    81   100   121   144
169   196
```

If there are fewer numbers to be printed than indicated by the FORM statement, an extra line feed is used in the FORM command.

For example, the desired result is to print a lower triangular matrix which would appear as

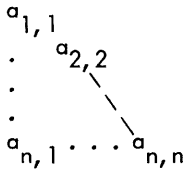


where the dimensions are N by N and the maximum element size may be ±99.99.

The output sequence in the program would be

```
>4.0  DO STEP 20 for I = 1 TO N (RET)
>20.0 TYPE IN FORM 5: A(I, J) FOR J = I UNTIL (LF)
      J > N OR J > I (RET)

>FORM 5: (LF)
         (LF)
  %%%.%% %%%.%%....%%%.%% %%%.%% (RET)
```

In addition, there must be N+1 fields specified by the FORM statement. At execution, the result would be



INPUT FROM FILES

The CAL language has the facility to accept input data from data files. A data file is one that has been created and defined previously by use of the executive mode, another subsystem, or by CAL itself. A data file differs from a program file, in that the program file contains the steps of a program, and a data file contains numbers to be read by a program.

To open a data file to read in CAL, the instruction is

OPEN "Name" FOR INPUT AS FILE (expression)

For example, assume a matrix $\overline{A}$ with row dimension M and column dimension N has been stored previously in a file named "MATRIX." The CAL statement to input this to a program might appear as

```
>1.0 OPEN "MATRIX" FOR INPUT AS FILE 999 (RET)
>1.1 DO PART 2.0 FOR I = 1 TO M (RET)
>1.2 CLOSE 999 (RET)
  :
  :
>2.0 DO PART 3.0 FOR J = 1 TO N (RET)
>3.0 READ FROM 999: A(I, J) (RET)
```

The general form of the READ statement is

READ FROM (expression): $v_1, v_2, \ldots, v_n$

In the example above, STEP 1.2 will close the file. The command is

CLOSE (expression)

At execution, this command will close the file; i.e., it is transferred from memory to the disc. The file then is unavailable for input until it is reopened.

Similarly, a file may be written on by using the "WRITE ON" statement. Consider the following example.

```
>1.0 OPEN "DATA" FOR OUTPUT AS FILE 1 (RET)
>2.0 DO PART 3 FOR I = 1 TO 100 (RET)
>2.5 TO STEP 4.0 (RET)
>3.0 WRITE ON 1: I (RET)
>4.0 CLOSE 1 (RET)
```

The general form for the WRITE ON command is

WRITE ON (expression): $e_3 \ldots e_2, \ldots, e_n$

After execution, the file called DATA would appear as

```
I = 1
I = 2
I = 3
  :
  :
```

If this file is subsequently read by a CAL program, the non-numeric characters would be ignored so that it would seem as though the file contained only 1, 2, 3, etc. However in the case of subscripted variables, the numerical value of each subscript is written on the file, so that if the file were to be read by a CAL program the subscripts would be mistakenly read as data. Consider the following program.

```
>1.0  OPEN "DATA" FOR OUTPUT AS FILE 1 (RET)
>2.0  DO PART 20 FOR I = 1 TO 5 (RET)
>3.0  TO STEP 22 (RET)
>20.0 DO PART 21 FOR J = 1 TO 5 (RET)
>21.0 DEMAND A(I, J) (RET)
>21.1 WRITE ON 1: A(I, J) (RET)
>22.0 CLOSE 1 (RET)
```

This program causes CAL to write the following on the file called DATA.

```
A(1, 1) = 1
A(1, 2) = 2
A(1, 3) = 3
A(1, 4) = 4
A(1, 5) = 5
A(2, 1) = 6
A(2, 2) = 7
A(2, 3) = 8
A(2, 4) = 9
A(2, 5) = 10
A(3, 1) = 11
A(3, 2) = 12
A(3, 3) = 13
A(3, 4) = 14
A(3, 5) = 15
A(4, 1) = 16
A(4, 2) = 1
A(4, 3) = 18
A(4, 4) = 19
A(4, 5) = 2
A(5, 1) = 21
A(5, 2) = 22
A(5, 3) = 23
A(5, 4) = 24
A(5, 5) = 25
```

To a CAL program, the file DATA would incorrectly appear as

1, 1, 1, 1, 2, 2, 1, 3, 3, etc.

To avoid this confusion, the user should use the WRITE ON statement with a form statement.

WRITE ON (expression) IN FORM (expression): $e_1 \ldots, e_n$

Thus, if statement 21.1 in the preceding program were changed to

```
>21.1 WRITE ON 1 IN FORM 1: A(I, J) (RET)

>FORM 1: (LF)
 %%%%%%% (RET)
```

the resulting file DATA would correctly appear to a CAL program as

1, 2, 3, 4, 5, 6, 7, . . . , 25

APPENDIX. CAL SUMMARY

NUMBERS

Integer (no decimal), e.g., 30000

Floating point (has a decimal part), e.g., 30000.00

Scientific notation, e.g., 30E3 (where E3 means 3 to the power of 10)

Numbers may be input as integers if they have less than 8 digits; otherwise, they must be in scientific notation.

VARIABLES

Examples of legal variables: A, B, C,...,Z; A(1), A(2),..., A(N); B(1), B(2),...B(N), etc.; A(1, 2), A(N, M); A((X(N)+3, A(M+1)).

Examples of illegal variables: A1, B1, AB, etc.

ARITHMETIC OPERATORS

In order of precedence:

- ↑ Exponentiation
- - Unary minus (e.g., -A↑2 = negative of A^2)
- *, / Multiplication, division
- +, - Addition, subtraction

LOGICAL OPERATORS

In order of precedence:

- = Equal to
- # Not equal to
- > Greater than
- < Less than
- NOT
- AND OR
- = Replacement

MATHEMATICAL FUNCTIONS

ABS(A)	Absolute value of A
SIN(A)	Sine of A
COS(A)	Cosine A
TAN(A)	Tangent of A
ATAN(X, Y)	Arctangent (X/Y) ($\pi > R > -\pi$) (the trigonometric functions take their arguments in radians or return results in radians)
EXP(A)	E to the power A
LOG(A)	Natural logarithm
LOG10(A)	Base 10 logarithm
SQRT(A)	Positive square root of A
IP(B)	Integer part of B (If B=246.25, IP=246)
FP(B)	Fractional part of B (If B=246.25, FP=.25)
SUM(A, B, C, D, ...)	Sums the list
PROD(A, B, C...)	Finds the product of the list
MAX(A, B, C, ...)	Finds the maximum in the list
MIN(X, Y, Z, A, B, ...)	Finds the minimum in the list
e MOD n	Modular arithmetic

COMMANDS (Direct or Indirect Statements)

Direct (with statement number) or indirect (without statement number):

SET V = e

TYPE $e_1, e_2, e_3, \ldots$

TYPE STEP n.n

TYPE PART e

TYPE ALL STEPS

TYPE ALL VALUES

TYPE ALL

TYPE IN FORM e: $e_1, e_2, \ldots, e_n$

TYPE FORM e

TYPE ALL FORMS

TYPE "string"

TO STEP n.n

TO PART e

DO PART n.n

DO PART e

PAGE (Causes CAL to skip a page)

LINE (Causes CAL to skip a line)

COMMANDS (Direct Statements Only)

RETURN e	(Used with multiple-step subprograms, i.e., function statements)
PAUSE	(Halts program and waits for instructions)
DONE	(Stops a DO PART)

DELETE v (v = any variable)

DELETE STEP n.n

DELETE PART e

DELETE ALL

INPUT FROM

OUTPUT TO

DUMP ⏎

TO/File Name/

LOAD ⏎

FROM/File Name/

EDIT STEP n.n

EDIT FORM e

DEFINE f $[v_1, v_2, v_3, \ldots]$ =e

(f is any variable name
v is any variable
e is any expression)

DEFINE f $[v_1, v_2, \ldots, v_n]$: statement

GO
STEP
CANCEL

MODIFIERS

IF e
UNLESS e
FOR v = e BY e TO e
WHILE e
UNTIL e

WHERE e
IF e THEN e ELSE e

EDIT CHARACTERS

A^c	Prints ↑ and deletes preceding character
W^c	Prints \ and deletes preceding word
Q^c	Prints ← and deletes preceding line
⏎	Throws away the rest of the old line and ends the edit.
C^c	Copies a character
S^c	Skips a character and prints %.
Z^cC	Copies up to character C, inclusive
X^cC	Skips up to character C, inclusive
R^c	Retypes
T^c	Retypes and aligns
Y^c	Copies rest of old line but does not end the edit.
D^c	Copies and prints out rest of old line and ends the edit.
F^c	Copies rest of old line but does not print it out; end the edit.
$E^c \ldots E^c$	Allows characters to be inserted between two points; first E^c prints a < while the second prints a > and returns user to edit mode.

SDS SCIENTIFIC DATA SYSTEMS • 1649 Seventeenth Street • Santa Monica, California 90404

EXECUTIVE OFFICES
1649 Seventeenth Street
Santa Monica, Calif. 90404
(213) 871-0960

DEVELOPMENT DIVISION
2525 Military Avenue
Los Angeles, Calif. 90064
(213) 879-1211

MANUFACTURING DIVISION
555 South Aviation Blvd.
El Segundo, Calif. 90245
(213) 772-4511

MARKETING DIVISION
1649 Seventeenth Street
Santa Monica, Calif. 90404
(213) 871-0960

SYSTEMS DIVISION
555 South Aviation Blvd.
El Segundo, Calif. 90245
(213) 772-4511

600 East Bonita Avenue
Pomona, Calif. 91767
(714) 628-7371

12150 Parklawn Drive
Rockville, Maryland 20852
(301) 933-5900

PROGRAMMING
2526 Broadway Avenue
Santa Monica, Calif. 90404
(213) 870-5862

INSTRUMENTS
555 South Aviation Blvd.
El Segundo, Calif. 90245
(213) 772-4511

TRAINING
1601 Olympic Boulevard
Santa Monica, Calif. 90404
(213) 871-0960

SALES OFFICES

EASTERN

Maryland Engineering Center
12150 Parklawn Drive
Rockville, Maryland 20852
(301) 933-5900

69 Hickory Drive
Waltham, Mass. 02154
(617) 899-4700

1301 Avenue of the Americas
New York City, N. Y. 10019
(212) 765-1230

673 Panorama Trail West
Rochester, New York 14625
(716) 586-1500

One Bala Avenue Building
Bala Cynwyd, Pa. 19004
(215) 667-4944

SOUTHERN

Holiday Office Center
Suite 4
3322 South Memorial Pkwy.
Huntsville, Alabama 35801
(205) 881-5746

Washington Plaza North
Suite 111
3880 Highway U.S. 1 South
Titusville, Florida 32780
(305) 267-6181

2964 Peachtree Road, N.W.
Suite 350
Atlanta, Georgia 30305
(404) 261-5323

8383 Stemmons Freeway
Suite 233
Dallas, Texas 75247
(214) 637-4340

3411 Richmond Avenue
Suite 202
Houston, Texas 77027
(713) 621-0220

MIDWEST

2720 Des Plaines Avenue
Des Plaines, Illinois 60018
(312) 824-8147

17500 W. Eight Mile Road
Southfield, Michigan 48076
(313) 353-7360

Suite 222, Kimberly Bldg.
2510 South Brentwood Blvd.
Brentwood, Missouri 63144
(314) 968-0250

Seven Parkway Center
Pittsburgh, Pa. 15220
(412) 921-3640

WESTERN

1360 So. Anaheim Blvd.
Anaheim, Calif. 92805
(213) 865-5293

2526 Broadway Avenue
Santa Monica, Calif. 90404
(213) 870-5862

505 W. Olive Avenue
Suite 300
Sunnyvale, Calif. 94086
(408) 736-9193

World Savings Bldg.
Suite 401
1111 So. Colorado Blvd.
Denver, Colo. 80222
(303) 756-3683

Fountain Professional Bldg.
9000 Menaul Blvd., N.E.
Albuquerque, N. M. 87112
(505) 298-7683

Dravo Bldg., Suite 501
225 108th Street, N.E.
Bellevue, Wash. 98004
(206) 454-3991

CANADA

864 Lady Ellen Place
Ottawa 3, Ontario
(613) 722-8387

INTERNATIONAL REPRESENTATIVES

FRANCE

Compagnie Internationale pour l'Informatique

EXECUTIVE OFFICES
101 Boulevard Murat
Paris 16ème

SALES OFFICES
17 Rue de la Reine
Boulogne 92

MANUFACTURING AND ENGINEERING
Rue Jean Jaures
Les Clayes Sous Bois 78

ISRAEL

Elbit Computers Ltd.
Subsidiary of Elron Electronic Industries Ltd.
88 Hagiborim Street
Haifa

JAPAN

F. Kanematsu & Co. Inc.
Central P. O. Box 141
New Kaijo Building
Marunouchi, Chiyoda-Ku
Tokyo

90 11 14A