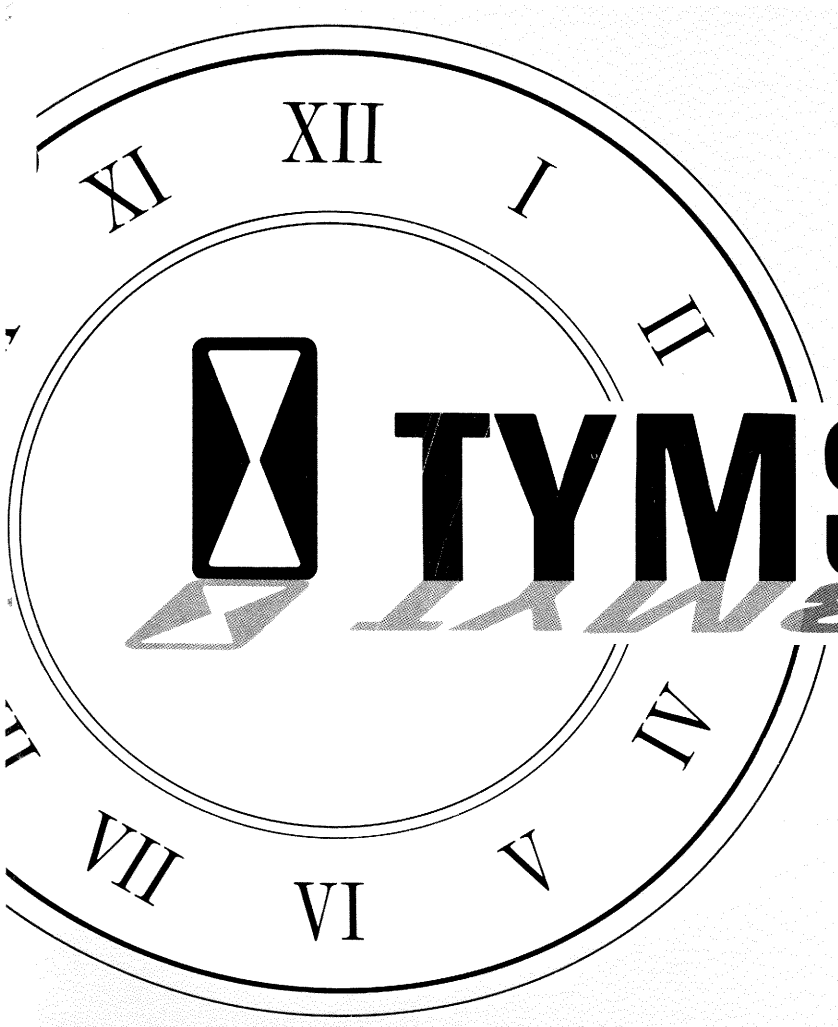
CAL
REFERENCE SERIES



TYMSHARE®

Price $3.50

TYMSHARE MANUALS

REFERENCE SERIES

CAL

June 1969

TYMSHARE, INC.
CORPORATE OFFICES
525 UNIVERSITY AVENUE, SUITE 220
PALO ALTO, CALIFORNIA 94301

DIVISION OFFICES

Los Altos, California ■ Inglewood, California ■ Englewood Cliffs, New Jersey

DISTRICT OFFICES

Seattle, Washington ■ Dallas, Texas ■ Newport Beach, California ■ Arlington, Virginia

TYMSHARE MANUALS
SYMBOL CONVENTIONS

The symbols used in this manual to indicate Carriage Return, Line Feed, and ALT MODE/ESCAPE are as follows:

Carriage Return: ↵

Line Feed: ↓

ALT MODE/ESCAPE: ⊕ *NOTE: This symbol will be printed as many times as it is required to hit this key.*

Action At The Terminal

To indicate clearly what is typed by the computer and what is typed by the user, the following color code convention is used.

Computer: Black **User:** Red

CONTENTS

PREFACE

The CAL Reference Manual is designed both as a tutorial and a reference text. The manual covers all of the CAL language facilities, both basic and advanced.

CAL (Conversational Algebraic Language) is a unique language that was designed especially for a time sharing environment. Unlike most of the older sequentially oriented languages such as FORTRAN, CAL is designed to operate procedurally as does the human mind. This feature makes CAL easy to use and work with. Each procedure or part may be debugged individually and then combined easily into a complete program. The unique characteristic of being able to execute parts of the program in non-sequential order makes CAL particularly good for solving problems which are interactive; problems in which the results of one series of calculations may be evaluated to determine the next series of calculations needed.

CAL has many features which are particularly designed for on-line, conversation use which many computer languages do not have. Included are

- Direct and indirect modes of execution.
- Means of editing a single statement or variable without recompiling in the ordinary sense.
- Being in the EDIT mode whenever commands are being written.
- Compiling each statement immediately after it is typed in and returning an error diagnostic if the statement is incorrect.

SECTION 1
INTRODUCTION TO CAL

As an introduction to CAL programming we have written a simple program which computes gas mileage when the initial and final odometer readings for any given amount of gasoline are known.

```
1.0 DEMAND I, F, G
2.0 T = F–I
2.1 M = T/G
3.0 TYPE T, M
```

Let's go back and see how this program was written. If you can understand the techniques used in writing this program you should have no trouble writing many other programs which use the same commands.

Defining The Problem

The first and probably most important step in programming is writing a clear, concise definition of the problem. A computer is designed to follow sets of simple commands which appear in logical sequence. It is during this first step that you should organize your problem into small sections that can be written in CAL.

The easiest way to define a problem for programming is to separate the problem into the following three sections.

1. OUTPUT - What information is desired? This section includes the answer to our problem and anything that we wish to have printed, in this case, the gas mileage.
2. COMPUTE - What computations must be made to find the above information?

   Gas Mileage = Total miles travelled divided by the amount of gas used.

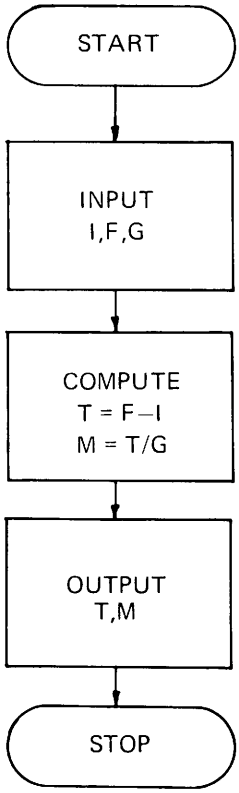
   Total Miles = Final odometer reading minus initial odometer reading.
3. INPUT - What information must be supplied to solve the problem?

   Initial odometer reading.

   Final odometer reading.

   Amount of gas used (in gallons).

Flowcharting The Problem



Entering The Computer And Calling CAL

The process of calling the computer and identifying yourself is called logging in.

After the connection has been properly made, the computer replies with:

PLEASE LOG IN: ↵ . . . . . Type a Carriage Return.

ACCOUNT: A3 ↵ . . . . . Now the user types in his account number A3 followed by a Carriage Return.

PASSWORD: ↵ . . . . . The user types his password followed by a Carriage Return. The letters in the password do not print on the page.

USER NAME: JONES ↵ . . . . . The user types his user name JONES followed by a Carriage Return.

PROJ CODE: K-123-X ↵ . . . . . The user has typed K-123-X as a project code. *NOTE: A project code is optional with the user. If no project code is wanted, simply type a Carriage Return in response to the system's request.*

READY 12/8 11:20 . . . . . The user may now call CAL.

–CAL ↵ . . . . . When the EXECUTIVE dash (–) appears, the user calls CAL by typing CAL and a Carriage Return. The system will reply with the question STATEMENTS =. The user must then type an estimate of the maximum number of statements that he will need for the program followed by a Carriage Return. If fewer than 50 statements are needed and no heading is desired, the user may press a Carriage Return at this point.

STATEMENTS = ↵

> 1.0 DEMAND I,F,G ↵ . . . . . Now the steps of the program are typed. Each step is preceded by a step number. The program will be executed in the order specified by the numbers. Typing a statement with the same number as a preceding statement will cause the preceding statement to be replaced. The first step contains the command DEMAND which, during execution, will request the values of I,F, and G from the user.

> 2.0 T = F–I ↵ . . . . . The arithmetic computations shown here are called replacement statements. The first replacement statement computes the total mileage and the second computes the gas mileage.

> 2.1 M = T/G ↵

> 3.0 TYPE T,M ↵ . . . . . This command will type the values of the variables T and M.

> RUN ↵ . . . . . We now begin execution of the program by using the command RUN.

I = 1125.7 ↵ . . . . . The first statement of the program DEMAND is now executed. The value of variable I is requested and the user types 1125.7 and terminates the value with a Carriage Return. In the same manner the values of F and G are supplied to the program.

F = 1764.1 ↵

G = 40.9 ↵

T = 638.40000000 . . . . . After computing the values of T and M the TYPE statement causes these values to be typed.

M = 15.60880200

> QUIT ↵ . . . . . Control is then returned to the user. Wishing to leave CAL, he types the command QUIT and returns to the EXECUTIVE.

```
– LOGOUT ↵
TIME USED 0:05:17
PLEASE LOG IN:
```

The EXECUTIVE command LOGOUT tells the computer that the user is finished. Then, the computer types the amount of time used, 5 minutes and 17 seconds in this case. The PLEASE LOG IN: command is repeated to allow another person to enter the system on that terminal. If there is no one waiting, hang up the phone.

CAL Statements

A CAL program is written in simple logical steps called statements. Approximately 200 statements are allowed per program, depending upon the length of each statement and the number of variables used. A CAL statement may be either a command statement, a FORM statement, or a DEFINE statement. A statement may be up to 256 characters long and is always terminated by a Carriage Return. If a statement is longer than one terminal line (72 characters), it may be continued by typing a Line Feed instead of a Carriage Return.

Statement Elements

CAL statements are composed primarily of combinations of CAL commands (such as TYPE) and constants (numeric values), and variables which are connected by arithmetic, logical, or relational operators. The following section defines the statement elements used in CAL.

Constants

Constants (numeric values) may be represented in three ways. They may be expressed as integers (whole numbers without a decimal point), as decimals (numbers used with a decimal point), or in scientific notation. Scientific notation is used with very large or very small numbers. For example, if we wish to input the number of miles travelled by a beam of light in a year we could use 6E12 instead of 6,000,000,000,000 ($6x10^{12}$). The E in this notation indicates that the number to the left of the E (which may be in integer or decimal form) is multiplied by 10 raised to the power of the number appearing to the right of the E (the exponent may be a positive or negative integer).

Certain size limitations on the numbers should be noted. CAL will accept numbers as large as 10^{77} and as small as 10^{-77}. However, internally CAL will retain only 11 places of significance (the numbers are rounded to 11 significant digits) and will output only up to 8 significant digits. CAL will output a number as an integer if it contains less than 9 integer digits, and as a decimal if it contains less than 8 integer digits and/or less than 9 decimal digits. If the number has more than 8 integer or 8 decimal digits, it will be output in scientific notation.

NOTE: FORM statements alter these rules, and will be explained on Page 10.

CAL Command	CAL Output
> TYPE 1123.456789	1123.45680000
> TYPE –1123.4567	–1123.45670000
> TYPE +.1234	0.12340000
> TYPE 30000000	30000000
> TYPE 30000000.00000	30000000
> TYPE 6666666666666666	6.66666670E 15
> TYPE .00123456789	1.23456790E–03
> TYPE 1234567890E25	1.23456790E 33
> TYPE .1234E–23	1.23400000E–24
> TYPE –5.9↑23	–5.36532790E 17
> TYPE .1234E04	1234
> TYPE 1234E–04	0.12340000

Variables

A variable is defined as any symbol (I, F, G) which represents a numeric or logical value which may change during execution of a program. Legal variables in CAL include any single letter (A-Z) and any single letter/single number combination within the range A0-Z9. In addition any legal CAL variable may be used with subscripts.

Subscripted variables appear in the following form: A(1), A(I,J), R2(3,N,X,B-3). Any legal CAL variable may be used with subscripts. CAL subscripts consist of any number of positive integers and/or single variables and/or expressions,[1] separated by commas, and enclosed in parentheses. A variable may have any number of subscripts. Each subscripted variable is treated as a unique variable.

1 - If an expression is used, it will be truncated to the nearest smaller integer and this integer taken modulo 2^{23} (MOD).

Expressions

In its simplest form an expression may consist of a single constant, a variable, or a function. An expression also may denote a computation between two or more constants and/or variables or functions.

There are two kinds of expressions in CAL: arithmetic and logical. The value of an arithmetic expression is always numeric. The value of a logical expression is either 1 (true) or 0 (false).

Arithmetic Expression

An arithmetic expression may consist of a single constant, variable, or function. More complicated arithmetic expressions may be formed by combining variables, constants, or functions using the arithmetic operators.

Arithmetic Operators And Precedence

Five different arithmetic operations may be performed in CAL.

Operation	Symbol	Precedence
Exponentiation	↑	Computed first
Division	/	Computed second
Multiplication	*	
Modulo The MOD operator computes the remainder which results from dividing the value or expression to the left of MOD by the value or expression to the right of MOD. For example, A MOD B is equivalent to the expression FP (A/B) *B.	MOD	
Subtraction	−	Computed third
Addition	+	
Replacement The replacement arrow assigns the value of the expression on the right to all of the variables listed. For example, A←B←C←0 would assign a value of zero to the variables A, B, and C.	←	Computed last

In general CAL follows the established arithmetic rules. CAL does require, however, that an operator always be specified. Thus, 4X is not a legal expression. In CAL it must be written 4*X.

NOTE: The negative sign (or unary minus) is evaluated before any arithmetic operations are performed. Thus −3↑2 is equal to +9 in CAL.

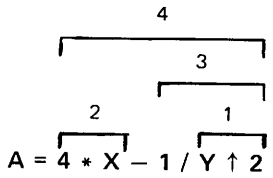
Precedence

In normal algebraic expressions, only one operation can be performed at a time. The computer also can perform only one operation at a time, and therefore since most expressions contain more than one arithmetic operator, some order or priority of computation (execution) must be established. The order of computation used in CAL is the same one found in simple algebra.

All arithmetic expressions in CAL are scanned from left to right. If any exponentiation is encountered, it is computed first. The system next checks for any * or / or MOD operations which are then executed from left to right. The system then will compute all + and −, again working from left to right, and last the replacement (←) operation is performed.

Example

The operators in the expression A = 4*X-1/Y↑2 would be executed in the following order:



we have $A = 4X - \frac{1}{Y^2}$

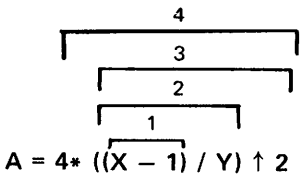
Parentheses

The normal order of execution may be altered by using parentheses in a CAL expression. Anything that appears in ( ) must be evaluated before the expression can be solved. The inner set of parentheses is always evaluated first.

Notice how parentheses in an equation alter the normal order of execution.

Example

A = 4 * ((X - 1) / Y)↑2 is executed starting with the inner set of ( ).



in this case $A = 4\left(\frac{X-1}{Y}\right)^2$

Logical Expressions

A logical expression may consist of a single logical constant or a logical variable. The value of a logical expression is always a truth value; that is, 1 if the expression is true, 0 if it is false.

More complicated logical expressions may be formed by using logical and relational operators. These expressions may be one of the following forms:

1. Relational operators combined with arithmetic expressions.
2. Logical operators combined with logical constants or logical variables.
3. Logical operators combined with either or both forms of the logical expressions described in 1. and 2. above.

Relational Operators

A relational operator makes a comparison between expressions. For example, the expression

X>Y

means X is greater than Y. This expression has a value which at any time is either true or false. Comparisons may be made using any of the following relational operators.

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

All of the relational operators have the same precedence and hence will be cleared starting with the leftmost operator.

For example, the expression R = 3 would have a value of 1 (true) if R is equal to 3, and a value of 0 (false) if R is equal to any other value.

The six relational operators may be combined by using a logical AND or OR. If an AND is used to connect two relational operators, both of the specified relationships must be true before the command can be executed. If the OR is used as a connector, only one of the relationships must be true.

For example, if R = 3 and S = 4 the entire expression R = 3 AND S = 5 would have a value of 0 (false); R = 3 is true but S = 5 is false, and hence the entire expression is false. On the other hand, the expression R = 3 OR S = 5 would have a value of 1 (true).

DIRECT And INDIRECT Statements

Most CAL statements may be used either indirectly (with a step number) or directly (without a step number). A direct (no step number) statement is executed as soon as the Carriage Return is pressed. Direct statements are not saved after execution and thus may not be reused. The following example uses direct statements:

```
> DEMAND A ↵          DEMAND command executed
        A = 34 ↵      immediately.
> X = A↑2 ↵           Replacement command executed immediately.
> TYPE X ↵            TYPE command executed im-
      X =   1156      mediately.
>
```

Indirect statements (with a step number) are not executed immediately. They are stored in a program in step number sequence. All indirect statements are referred to by their step number. When program execution is started (using a direct TO control command or the direct RUN command), the computer will step through the entire set of indirect statements in the specified sequence. The entire program (all of the indirect statements) will be saved after execution. Thus by using another direct RUN command you can execute the program again.

In the following example the indirect statements are stored in a program which is then executed twice using two direct RUN commands.

```
> 1.0 DEMAND A ↵      Stored
> 1.1 X = A↑2 ↵       Stored
> 2.0 TYPE X ↵        Stored
> RUN ↵
        A = 34 ↵      DEMAND command executed.
      X =   1156      TYPE command executed.
> RUN ↵
```

```
A = 122 ↵      DEMAND command executed again.
X =   14884    TYPE command executed again.
>
```

NOTE: Every indirect statement must have a unique step number. If two statements are typed with the same step number, the second statement will replace the first.

Step And Part Numbers

Step numbers are used to number indirect statements. Step numbers may be either integer or decimal numbers, and may range from 0 to 999.999. Part numbers are the integer part of step numbers. Part numbers may range from 0 to 999. All steps (statements) with the same part number are said to belong to the same part and may be referred to as a group.

SECTION 2
INPUT AND OUTPUT

The input and output commands are presented here in four sections. The first section includes the simple DEMAND and TYPE commands used in the sample program. The second section introduces the FORM statement which allows programmer formatted input and output. The third section introduces input and output of literal text. The last section presents a list of output commands used primarily to display the program. CAL also allows data file input and output. Data file input/output is covered in the special section on data files, Page 23.

Simple Input/Output

DEMAND Command

To input a number from the terminal, CAL uses the command DEMAND with a list of variables separated by commas. When the DEMAND command is executed, the computer will type the variable name and then wait until a value is supplied by the user.

Example

```
> DEMAND A              DIRECT command
    A =
```

Typing Numeric Data Into Variables

When CAL demands a numeric value to fill a variable, simply type the number. If you wish the variables specified in the DEMAND statement to be typed on separate lines, press the Carriage Return after you type in each numeric value. If you wish the next variable to be typed on the same line, press the space bar, or type a comma or a semicolon after each variable.

If you make a mistake, type a Control W (W^C), and then retype the entire number correctly. If you type in any non-numeric characters the computer will normally return a ?, and you then must retype the entire number. Non-numeric characters typed before numeric values are ignored.

TYPE Command

The CAL TYPE command, when executed, will output the value of the variable and/or expression (A, B+4, A/B) used with it. If more than one variable or expression is used, they must be separated by commas.

```
> DEMAND A,B
        A = 5.302       B = 7.963
> TYPE A,A+A*B,(3+4-7*A)/8
        A =             5.30200000
    A+A*B =            47.52182600
(3+4-7*A)/8 =          -3.76425000
>
```

Programmer Formatted I/O

CAL allows the individual programmer to input and output his data in any format and with any text he desires. The format for the input or output is specified in a FORM statement. A FORM statement is not an actual command; it merely specifies the format to be used and will be ignored entirely during execution of the program unless called by (used with) a special input (DEMAND) or output (TYPE or WRITE) command.

If, for example, we are listing the amount and cost of gasoline per tankful, it might be nice to have it printed in the following form:

```
 8.5 GAL.    $3.21
10.6 GAL.    $4.01
etc.         etc.
```

rather than in the normal CAL format.

```
G =  8.5     C = 3.21
G = 10.6     C = 4.01
```

To input information in his own form, the user must use two separate CAL statements as follows:

```
(1)  1.0 DEMAND IN FORM 1:G,C
(2)  FORM 1:
         # GAL.          $ #
```

The first statement in the example is an input command. It specifies the form (calling it by its form number) and lists the variables to be used in the form in the order in which they occur.

CAL will recognize only positive integer form numbers. If a non-integer or an expression is used as the form number, CAL automatically will take the integer part (see Standard Functions) of the expression or non-integer, thus converting it to an integer. If the form number is negative, CAL will not accept it and will give an error message.

The general form for the input or output command used with a FORM statement is

DEMAND IN FORM (form number): The variables listed in the order of occurrence

The output commands TYPE and WRITE (data files) may be used in place of the DEMAND.

Form Statements

The second and third lines of the example are the FORM STATEMENT

```
FORM 1: ↵
     # GAL          $ # ↵
```

A FORM statement must appear on at least two lines. The first line specifies the form number followed by a colon, terminated with a Line Feed. A form is referred to by its form number. *CAUTION: Never use a step number with a FORM statement.*

All subsequent lines in a FORM statement contain the form specification which lays out the format in which the input or output is to appear. Input and output form specifications are very different and may not be interchanged.

Form Specifications (Input)

Input form specifications use a single # to indicate a variable in the layout. Any type of text may be put in a form specification.

Form Specifications (Output)

The output form specification is more complicated since the user must specify the maximum number of digits that will be printed and the type of number to be printed (integer, decimal, or scientific notation). As with the input form specification, any text may be used in the output form specification.

Integer Output

To specify integer output, a % sign must be typed for each digit in the variable. If there is any chance that the variable will be negative, an extra % sign must be specified. If the variable is not an integer, it will be rounded to an integer, dropping any decimal places when it is printed. *NOTE: Only eight integer places are allowed in CAL. The maximum size of an integer numeric field specification is 9 places (9 % signs).*

Decimal Output

Decimal output also uses a % sign to specify the number of digits to be printed. With decimal output however, a decimal point must be specified. For example, the numeric field specified by %%%%.%% will print up to four integer places and two decimal places. As with integer output, if there is any chance that the number will be negative, an extra % sign must be specified. Up to 16 places (16 % signs) may be specified with decimal output; however, the output will be rounded to 8 significant digits.

Exponential Output (Scientific Notation)

If the variable is longer than 8 integer places or 8 decimal places, the output must be specified in exponential form. With scientific notation a # is used to specify each place needed. A minimum of 6 #'s and a maximum of 15 #'s may be specified. The minimum form includes a place for:

1. A positive sign (or blank), or a negative sign to indicate the sign of the number.
2. One integer digit.
3. The E.
4. A blank or negative sign to indicate the sign of the exponent.

5 and 6. Two places for the exponent.

Example

```
>1.0 DEMAND IN FORM 1: A ↵
>FORM 1: ↵
        A IS THE ORIGINAL NUMBER. IT IS SUPPLIED AT THIS POINT ↵
                    A = # ↵
>2.0 TYPE IN FORM 12: A,A,A, ↵
>FORM 12: ↵
        FOR INTEGER OUTPUT          %%%%%% ↵
        FOR DECIMAL OUTPUT          %%%%%%.%%%% ↵
        FOR EXPONENTIAL OUTPUT      ############ ↵

> TO PART 1 ↵
        A IS THE ORIGINAL NUMBER. IT IS SUPPLIED AT THIS POINT
                    A = 1234.5878 ↵

        FOR INTEGER OUTPUT          1235
        FOR DECIMAL OUTPUT          1234.5878
        FOR EXPONENTIAL OUTPUT      1.23459E 03
```

NOTE: If fewer numeric fields are specified than variables, the FORM statement will be repeated until all of the variables have been output. For example,

```
> 1.0 A = 1234 ↵
> 1.1 TYPE IN FORM 1:A,A,A ↵
> FORM 1: ↵↓
  %%%%          %%%% ↵
> RUN ↵
  1234          1234
  1234
```

Literal Text

Literal text may be inserted into a program in two different ways. It is used for comments, headings, descriptions, reminders, in short, anything the user wishes to remember.

TYPE "text"

This command may be used either directly or indirectly. Any combination of characters may be placed between the quote marks. When the program is executed everything within the quote marks will be printed. If the text is longer than 256 characters, it may be continued by using another TYPE " " statement. The TYPE " " also may be used to type headings on lists of data. See sample problems.

Comments

To add a comment to your program, type a !, then your comment, and a Carriage Return. Comments are used primarily for program explanation and documentation and may be inserted at any point in the program. **Comments are not printed during execution.** Comments may follow any indirect statement or they may occupy a separate line.

Display Commands

Output commands used to display the program are:

TYPE STEP step number	**TYPE ALL STEPS**
TYPE PART part number	**TYPE ALL FORMS**
TYPE FORM form number	**TYPE ALL FUNCTIONS**
TYPE function name	**TYPE ALL VALUES**
	TYPE ALL

The above commands are used to display an entire program or a specified portion of a program. These TYPE commands will type on the terminal the step, part, form, function (DEFINE statement), or values specified. These commands may be used both directly and indirectly.

Example: Literal Text In A Program

```
> 1.0 TYPE "THIS PROGRAM COMPUTES THE AREA OF 3 CIRCLES" ↵
> 1.1 A(R) = PI*R↑2 FOR R=5 BY 5 TO 15          ! R=5, 10, 15 ↵
> 1.2 ! A FOR MODIFIER IS USED TO SET THE VALUES OF R ↵
> 1.3 ! THE STANDARD FORMULA FOR AREA IS USED ↵
> 1.4 TYPE "      RADIUS              AREA"  !THIS IS A HEADING ↵
> 1.5 TYPE A ↵
> RUN ↵
THIS PROGRAM COMPUTES THE AREA OF 3 CIRCLES
     RADIUS            AREA
     A(5)=          78.53981600
    A(10)=         314.15927000
    A(15)=         706.85835000
```

SECTION 3
REPLACEMENT COMMAND

Replacement commands are merely arithmetic computations. Replacement commands appear in the following form:

a simple variable = any arithmetic expression

In the example in Section 1 (the gas mileage problem) there are two replacement commands. The first one computes the total mileage.

2.0 T = F - I

The second replacement command computes the gas mileage:

2.1 M = T/G

The term replacement command is used because the computer actually replaces the value stored in the variable on the left (T) with the computed value of the arithmetic expression (F - I). The equal sign (=) should not be read as "T is equal to", but as "the value stored in T is replaced by". In later examples we shall run into replacement commands that seem logically incorrect if the equal sign is interpreted to mean equality, such as T = T+1, but which make sense if the statement is read T "is replaced by" T+1.

SECTION 4
MODIFIERS AND CONDITIONS

This section includes the complete set of CAL modifiers and conditions. They may be used to modify all of the CAL commands unless otherwise specified. The CAL modifiers and conditions are used in the following ways:

- Cause commands to execute or not to execute depending upon the conditions specified by the modifiers; IF, UNLESS.
- Specify under which conditions execution of the command is to be terminated; WHILE, UNTIL.
- Cause the command to execute repeatedly (loop) over a specified range of values; FOR.
- Initialize or reinitialize variables; WHERE.

IF And UNLESS

The IF and UNLESS modifiers define the conditions under which the command modified will be executed. Any CAL command modified by an IF or an UNLESS will be executed if and only if the conditions specified by the modifier are met.

> 1.5 TO PART 5 IF R=20

The IF modifier used in this command will cause control to be transferred to PART 5 only if R=20; otherwise the command will not be executed.

In the statement

> 1.5 TO PART 5 UNLESS R=20

control will be transferred to Part 5 unless R=20. The command will not be executed if R=20.

> TYPE H IF H>=50 AND H<=75

The IF modifier in this command will limit the range of numbers typed from 50 through 75 inclusive.

```
>1.0 DEMAND H
>1.1 TYPE H↑2 IF H>=50 AND H<=75
>RUN
     H = 45          H is not within the specified range so H↑2 is not typed.

>RUN
     H = 55          H is within the specified
H↑2 =      3025      range so H↑2 is typed.
```

UNTIL And WHILE

The UNTIL and WHILE modifiers specify the conditions under which execution of the command modified is to be terminated. Any command modified by an UNTIL or a WHILE modifier will repeat (loop) until or while the conditions specified are met.

Example

DO PART 90 UNTIL T>=86

An UNTIL modifier used with a DO PART will cause repeated execution of the part until the condition specified; that is, T>=86, is met.

Example

```
>A=0
>TYPE (A←A+1)↑2 WHILE A<=5
  (A←A+1)↑2 =          1
  (A←A+1)↑2 =          4
  (A←A+1)↑2 =          9
  (A←A+1)↑2 =         16
  (A←A+1)↑2 =         25
```

FOR

The FOR modifier causes the command it modifies to be executed repeatedly over a range of values. A statement which repeats itself in this manner is said to loop, and thus the FOR modifier causes a command to loop. Four forms of the FOR modifier will be discussed.

The first form of the FOR modifier assigns specific values to the variables used. In the following example the FOR modifier assigns the values 2, 3, and 5 to the variable A and the command TYPE A↑2 is executed.

```
>TYPE A↑2 FOR A=2,3,5
     A↑2 =    4
     A↑2 =    9
     A↑2 =   25
>
```

The second form of the FOR modifier causes the command to loop for a specified range of values using a specified interval. The form used is as follows:

FOR variable = limit BY interval TO limit

```
>TYPE A FOR A = 0 BY 25 TO 100
```

If the desired interval is 1, specifying the interval is optional. The following two statements would produce the same results.

```
TYPE X FOR X = 1 TO 4
TYPE X FOR X = 1 BY 1 TO 4
```

The third and fourth forms of the FOR modifier are similar to the second except that execution of the command is terminated by a WHILE or an UNTIL. As soon as the conditions specified in the WHILE or UNTIL modifiers are met, the loop is considered completed. The general format is:

FOR variable = limit BY interval { **WHILE** / **UNTIL** } **condition**

Examples

```
>TYPE A↑2 FOR A = 1 BY 1 WHILE A↑2<=50
>TYPE A↑2 FOR A=1 UNTIL A↑2>50
```

General: The limits and interval may be an expression; for example, FOR X=X+1 BY R/X TO X↑2. It is possible also to use a negative interval such as FOR X=10 BY -2 TO 0. However, if the terminating condition is reached the first time through the loop (if the lower limit is already greater than or equal to the upper limit when the statement is executed), the statement will not execute.

NOTE: A FOR modifier may never be used with the control commands TO and DONE.

NOTE: If a FOR loop is used in a replacement statement, the results of the calculations usually are stored in subscripted variables. If a single variable is used, only the last computed value will be saved and printed because each time the command loops the old value stored in the variable is replaced by the new value.

In the following example two forms of the FOR modifier have been used.

```
>1.0 Z(A,B) = A*B FOR A = 1,2 FOR B = 10 BY
                                    10 TO 50
>1.1 TYPE Z
>TO PART 1
  Z (1,10) =    10      A = 1     B = 10
  Z (2,10) =    20      A = 2     B = 10
  Z (1,20) =    20      A = 1     B = 20
  Z (2,20) =    40      A = 2     B = 20
  Z (1,30) =    30      A = 1     B = 30
  Z (2,30) =    60        .         .
  Z (1,40) =    40        .         .
  Z (2,40) =    80        .         .
  Z (1,50) =    50        .         .
  Z (2,50) =   100        .         .
```

NOTE: The command TYPE Z will print the variable Z and all subscripted variables Z(0) through Z (X,Y,N. . .).

WHERE

The WHERE modifier is used to initialize variables that would normally have been set by replace-

ment commands. The general form of the WHERE modifier is:

WHERE variable = expression & variable = expression

The WHERE modifier may be used to modify a statement or an expression. If it is used to modify an expression, it should be used immediately following that expression. If it is used to modify the entire statement, it should appear at the end of the statement and must be separated from the statement by a comma. A WHERE statement modifier is read only once; a WHERE expression modifier is read each time the expression is evaluated. It is important that this difference be understood. The following example approximates the cube root of the number N. Note that the first WHERE modifier in the program is an expression modifier; it reinitializes the approximation until the difference between the old and new approximation is less than 10E-8. The second WHERE modifier is a statement modifier. It contains the first approximation of the cube root and will be read only once.

```
>1.0 DEMAND N
>1.2 B= (2*A↑3+N) / (3*A↑2) WHERE A=B UNTIL
        ABS (A-B)<10E-8, WHERE A=N/3 & B=N
>1.3 TYPE B
>RUN
        N=      81
                        B=    4.32674870
```

If we were to try to run this program leaving out the comma, thus making the statement modifier an expression modifier, the program will go into an infinite loop in step 1.2 because the terminating condition ABS(A-B) is never satisfied since A is reinitialized to the first approximation each time the statement is repeated.

IF. . .THEN. . .ELSE

The IF–THEN–ELSE modifier actually replaces a single expression in a command. The general form of the modifier is as follows:

IF $\overset{\text{condition}}{\underset{\text{list}}{}}$ THEN expression ELSE IF $\overset{\text{condition}}{\underset{\text{list}}{}}$ THEN expression ELSE expression

Any number of IF–THEN–ELSE's may be strung together to produce a resultant expression. An IF–THEN–ELSE modifier may replace any CAL expression. The modifier may end without an ELSE if desired; in which case the expression will result in a value of zero if none of the conditions are met.

Examples

```
1.0 A = IF X>0 THEN +1 ELSE IF X<0 THEN -1
                                    ELSE 0
4.5 TYPE IN FORM IF A<72 THEN 1 ELSE
                                2:A,B,C,D,(I)
3.7 DO PART IF X = 0 THEN 20 ELSE 30 FOR
                                R = 1 TO 10
1.5 R = IF X>0 THEN R↑2 ELSE IF X<0 THEN
                            ABS(R/2) ELSE R+1
>1.0 N = IF X MOD 2 = 0 THEN +1 ELSE IF X>10
                                THEN 0 ELSE -1
>1.1 TYPE N
>DO PART 1 FOR X= 4,12,13,5
        N =          1
        N =          1
        N =          0
        N =         -1
```

SECTION 5
CONTROL COMMANDS

The control commands control execution of the program. They are used to start and stop execution, and interrupt the normal order of execution. Programs are normally executed in step number sequence.

TO PART

The TO PART command is used directly to start execution of a program and indirectly to interrupt the normal order of execution. The TO PART command may be used with a part number, a step number, a predefined variable, or an expression. When used with a part number, the TO PART command transfers control to the lowest numbered step in the specified part. When used with a step number, the TO PART command transfers control to that step if it exists or to the next higher step. An expression used with a TO PART will be evaluated to eight places and considered as a step number. *NOTE: Step numbers may not be negative.* TO PART may be used with any of the CAL modifiers except FOR.

TO STEP

The TO STEP command may be used directly to start execution of a program or indirectly to interrupt the normal order of execution. A TO STEP command must be used with a specific step number and that step number must exist in the program.

Example

```
>1.0 A=13
>1.1 A=A+1
>1.2 TYPE A
>1.3 TO STEP 1.1 IF A#16
>TO PART 1
A=  13
A=  14
A=  15
```

DO PART

If the DO PART command is used directly, it executes only the part specified and then returns control to the user with a >. If DO PART is used indirectly, it interrupts the normal order of execution, executes the part specified, and then returns control to the step following the DO PART. DO PART is normally used with a part number; however, it may be used with a variable or an expression. *DO PART is commonly used with a FOR modifier to cause an entire part to loop.*

Example

```
>1.0 DO PART 2 FOR I=1 TO 4
>1.1 TO PART 3
>2.0 A(I) = I
>2.1 TYPE A(I)
>3.0 TYPE "THE END"
>TO PART 1
      A(1) =        1
      A(2) =        2
      A(3) =        3
      A(4) =        4
THE END
>
```

DO PART may be used with any of the CAL modifiers. The only restriction placed on the command is that control may not be transferred directly to a TO PART with a DO PART. Note that the DO PART is completed when the last step of **any** part is executed. Thus, if a part being called by a DO PART contains a TO control command, the part being done will be considered complete when the last step of **any** part is completed.

Example: Using a TO PART in the part called by a DO PART

```
>1.0 DO PART 2
>1.1 TYPE X
>1.2 PAUSE
>2.0 X=1
>2.1 TO PART 3
>2.2 X=X+1
>3.0 X=X+1
>3.1 TYPE "ALL"
>4.00 X=X+1
>4.1 TO STEP 2.0
>RUN
ALL
            X =             2
PAUSE IN STEP 1.2:
>
```

CAUTION: An indirect DO PART transfers control back to the step following the DO PART and then proceeds through the program. What would have happened if the PAUSE had not been specified after the DO PART?

DONE

Sometimes it is desirable to terminate the execution of a part called by a DO PART before the end of that part is reached. The DONE command terminates one loop of a DO PART command and returns control to the user if a direct DO PART was used, or to the DO PART to test for a terminating condition if an indirect DO PART was used. DONE is always used indirectly. It is generally used with a modifier to terminate execution of the part under specified conditions. DONE may be used with any modifier except FOR, UNTIL, and WHILE.

Example

```
>1.0 DEMAND A
>1.1 DO PART 3
>1.2 TO PART 4
>3.0 X=A↑2+4*A
>3.1 DONE IF X>100
>3.2 TYPE A,X
>3.3 A=A+1
>3.4 TO PART 3
>4.0 TYPE "THE END"
>RUN
        A = 6
        A =          6
        X =         60
        A =          7
        X =         77
        A =          8
        X =         96
THE END
>
```

DO STEP

The DO STEP control command is used directly to execute a single step. DO STEP is used indirectly to interrupt the normal order of execution, execute the step specified, and then resume normal execution at the step following the DO STEP. DO STEP always must be used with a specific step number which must exist in the program. Remember, the step specified may not be a TO control command. DO STEP may be modified by any of the CAL modifiers.

STEP

The STEP command always is used directly. It executes the next statement in the program and then returns control to the user. The STEP command is similar to the direct DO STEP except that the step number is implied.

PAUSE

The PAUSE control command stops execution of the program, gives a message indicating at what step the pause occurred and then returns control to the user. A PAUSE always must be used indirectly (with a step number). The program may be restarted at the step immediately following the PAUSE using a GO or TO PART control command. PAUSE can be modified by all modifiers except FOR, UNTIL, and WHILE.

Example

```
> 1.0 TYPE "COMPUTE THE SQ.RT. OF A"
> 1.1 DEMAND A
> 1.2 TO PART 2 IF A>0
> 1.3 TYPE "A MUST BE A POSITIVE NUMBER
                        NOT EQUAL TO ZERO"
> 1.4 PAUSE
> 2.00 TYPE A↑(1/2)
> 2.1 TO STEP 1.1
> RUN
COMPUTE THE SQ.RT. OF A
     A = 33
A↑(1/2) =                5.74456260
     A = -88
A MUST BE A POSITIVE NUMBER NOT EQUAL
                                TO ZERO
PAUSE IN STEP 1.4:
>
```

GO

The direct command GO is used to restart programs which have been interrupted at the point at which they were interrupted. A program may be interrupted during execution either by pressing an ALT MODE/ESCAPE or by inserting a PAUSE command in the program. In either case CAL remembers where the interruption occurred and the program may be restarted at this point by typing GO. If at any time CAL responds with NO GO it indicates that CAL does not know where to restart. In this case the user must use a TO or DO command to restart the program.

RUN

The direct command RUN starts execution of the program at the lowest numbered step in the program. RUN may be used interchangeably with the TO control commands to start execution of the program.

QUIT

The direct command QUIT transfers the user out of CAL to the EXECUTIVE. The user then may use the EXECUTIVE command CONTINUE to continue in CAL as long as he has not called any other language from the EXECUTIVE.

ALT MODE/ESC

Pressing this key interrupts whatever is going on at the moment. If the key is depressed during execution of a program, the program is interrupted and a message is typed specifying the step interrupted. If the key is pressed while the user has control, control will be returned to the EXECUTIVE. To return to the EXECUTIVE, it is better to use the QUIT command. If too many ALT MODE's are used and unwanted return to the EXECUTIVE occurs, type CONTINUE ↵ to return to CAL.

SECTION 6
FUNCTIONS

Functions are used to store sets of common computations so that they may be reused easily merely by specifying a function name. CAL has three types of functions: standard functions, iterative functions, and programmer defined functions.

Standard Functions

The standard functions are commonly used relationships which have been implemented in CAL as a convenience to the user.

PI

This constant contains the mathematical value of pi to eight significant digits.

```
> TYPE PI ↵
      PI =              3.14159270
> 1.0 DEMAND R ↵
> 1.1 TYPE PI*R↑2 ↵
> TO PART 1 ↵
          R = 22.5 ↵
  PI*R↑2 =              1590.43130000
```

SIN

This function computes the standard trigonometric function. The argument is the variable, number or expression contained within the parentheses in the following example. The argument must be in radians.

```
> TYPE SIN(90) ↵
  SIN(90) =             0.89388666

> TYPE SIN(PI-2.156) ↵
  SIN(PI-2.156) =       0.83359960
```

COS

This function computes the standard trigonometric cosine. The argument must be in radians.

```
> TYPE COS(90) ↵
  COS(90) =             -0.44807362

> X=COS(90) + SIN(90) ↵
> TYPE X ↵
      X =               0.44592305
```

TAN

This function computes the standard trigonometric tangent. The argument must be in radians.

```
> TYPE TAN(60) ↵
TAN(60) =               0.32004039

> TYPE TAN(87-45+32/9)*2 ↵
TAN(87-45+32/9)*2 =             -812.32034000
```

ATAN

This function computes the standard trigonometric arctangent. The answer will be in radians. The argument may be given in one of the three forms: ATAN (Y/X), ATAN(Z) where Z = Y/X, or ATAN(X,Y).

```
> TYPE ATAN(30/20) ↵
ATAN(30/20) =           0.98279372
> TYPE ATAN(1.5) ↵
ATAN(1.5) =             0.98279372
> TYPE ATAN(20,30) ↵
ATAN(20,30) =           0.98279372
```

LOG

This function computes the natural logarithm (log e) of an expression.

```
> TYPE LOG(2.17) ↵
LOG(2.17) =             0.7747217
> TYPE LOG(1+4/5*SIN(PI)) ↵
LOG(1+4/5*SIN(PI)) =            1.36424210E-12
```

LOG10

This function computes the decimal logarithm of an expression.

```
> TYPE LOG10(111) ↵
LOG10(111) =            2.04532300

> TYPE LOG10(34*6789/89) ↵
LOG10(34*6789/89) =             3.41389470
```

EXP

This function computes e (the base of the natural logarithm) raised to a power (any expression).

```
> TYPE EXP(3↑3) ↵
EXP(3↑3) =              5.32048240E 11

> TYPE EXP(0.77472717) ↵
EXP(0.77472717) =       2.17000000
```

ABS

This function computes the absolute value of an expression.

```
>TYPE ABS(-22.9)
ABS(-22.9) =          22.90000000
```

SQRT

This function computes the square root of an expression.

```
>TYPE SQRT(ABS(-25))
SQRT(ABS(-25)) =    5
```

IP

This function computes the integer part of the argument using the equation IP(X)=Y where Y is an integer and Y≤X. Note the treatment of negative arguments.

```
>TYPE IP(7.98765)
IP (7.98765) =                7

>TYPE IP(-2.579)
IP(-2.579) =                 -3
```

FP

This function computes the fractional part of the argument using the equation FP(X)=X-IP(X). Again note the treatment of negative arguments.

```
>TYPE FP(7.98765)
FP(7.98765) =                 0.98765000

>TYPE FP(-2.579)
FP(-2.579) =                  0.42100000
```

Iterative Functions

The iterative functions (SUM, PROD, MAX, MIN) compute a predefined set of computations. Iterative functions may be used by specifying the function name, the expression on which the function is to be computed, and the values to be used in the expression. The general form of the iterative functions is as follows:

Function name (implicit FOR clause:arithmetic expression)

The value of the expression is computed for each value given in the implicit FOR clause (a FOR clause without the FOR). The function is then computed using these expression values.

SUM

The SUM iterative function computes the value of the expression for each value specified in the implicit FOR clause and then adds all of these values. The SUM function is useful in integrating equations. The following example adds all of the even numbers from 12 to 24.

```
>TYPE SUM (C=12 BY 2 TO 24:C)
SUM (C=12 BY 2 TO 24:C) =          126
```

PROD

PROD computes the value of the expression for each step in the implicit FOR clause and then multiplies all of the computed values. In the following example the product function is used to compute the factorial of six.

```
>TYPE PROD(F=1 TO 6:F)
PROD(F=1 TO 6:F) = 720
```

MAX

MAX returns the largest value of the expression over the range of values given in the implicit FOR clause. In the following example the MAX function is used to determine the largest value of Y along a segment of a curve represented by the equation $Y=X^2-4X$ where X has values ranging from -1 to 6.

```
>TYPE MAX(X=-1 TO 6:X↑2-4*X)
MAX(X=-1 TO 6:X↑2-4*X) = 12
```

MIN

MIN returns the smallest value of the expression over the range of values given in the implicit FOR clause. The following example uses the MIN function to select the smallest value stored in T(J).

```
>T(J)=J↑2-12*J FOR J=1 TO 6
>TYPE MIN(J=1 TO 6:T(J))
MIN(J=1 TO 6:T(J)) = -36
```

NOTE: The implicit FOR clause may take any of the following forms:

(FOR) I=1,2,3
(FOR) I=1 TO 10
(FOR) I=0 BY 2 TO 50
(FOR) I=3 BY 7 UNTIL (I-4)↑2>50
(FOR) I=11 BY 4 WHILE I↑2-5<=50

AND and OR may not be used. If more than one variable is desired, the functions must be nested, such as:

A = SUM(X=1,2:SUM(Y=1 TO 6:X↑2+Y↑2))

This example would compute the SUM of the expression (X↑2+Y↑2) for X=1,2 and Y=1 to 6.

Programmer Defined Functions

The CAL programmer also is given the option of defining his own functions. Any computation or set of computations may be placed in a function. A function gives a computation a name and establishes its input and output parameters. After the function has been defined the programmer may use the computations simply by referring to the function name and listing the input and output parameters.

DEFINE

A function is defined with a DEFINE statement. A DEFINE statement is similar to a FORM statement in that it is not a command but simply a reference statement. Like a FORM statement, a DEFINE statement is never used with a step number. A DEFINE statement is referred to by the function name (in the following example as F). Two forms of the DEFINE statement exist: a short form used with a single calculation, and a longer form used with sets of calculations.

The short form appears as follows:

DEFINE $\overset{\text{function}}{\text{name}}$ [$\overset{\text{parameter}}{\text{list}}$] = **computation**

In the following example the function F is used with the parameters A and B:

DEFINE F[A,B] = SQRT (A↑2+B↑2)

Any non-subscripted variable that is legal in CAL may be used as a function name; however, a variable used as a function name may not be used at any other point in the program.

If more than one computation is desired in the function, the computations should be placed in step number sequence and a TO control command used in the DEFINE statement to transfer control to the step number of the first computation. The general form is as follows:

DEFINE $\overset{\text{function}}{\text{name}}$ [$\overset{\text{parameter}}{\text{list}}$]**:TO STEP** $\overset{\text{step or part}}{\text{number}}$

When this form of the DEFINE statement is used an **indirect** RETURN command must be used to terminate the function and return the computed value of the function to the point at which it was called. The general form of the RETURN command is as follows:

$\overset{\text{step}}{\text{number}}$ **RETURN variable**

A RETURN command may be used with a single expression; however, normally the computed value of the function will be stored in a single variable and that variable returned; that is, used with the RETURN command. *NOTE: A variable used as a parameter in the DEFINE statement may never be used with a RETURN command.*

Function Parameters

The function parameters may be specified by any non-subscripted variable. The variables used as function parameters are local variables; that is, they apply only to the function and do not exist outside of the function. Any variables used in the function which are not function parameters; that is, do not appear in the parameter list following the DEFINE statement, are taken to be global variables. Global variables apply to the entire program and contain the same value both when used in the function and when used outside of the function.

Using a Programmer Defined Function

A programmer defined function may be called simply by specifying the function name and then in brackets the values of the parameters. The parameter values may be specified using either actual numeric values or variables that have been defined previously in the program.

Example

```
> DEFINE F[A,B,X,Y]: TO PART 4      Function Defined
> 1.0 DEMAND A,B,C,D
> 1.2 TYPE ALL VALUES
> 1.3 S = F[5,5,5,5] +5             Function Called
> 1.4 TYPE ALL VALUES
> 1.5 TO PART 5
```

```
>4.0 TYPE ALL VALUES
>4.05 R = A↑2
>4.1 C = C+1
>4.15 X = X/2
>4.2 A = A+1
>4.25 TYPE ALL VALUES
>4.4 Z = A+B+C+X
>4.5 RETURN R
>5.0 TYPE "END"

>TO PART 1
```

A = 2	B = 2	C = 2	D = 2	*Step 1.0*
A =	2			*Step 1.2*
B =	2			*All global*
C =	2			*variables*
D =	2			
A =	5			*Step 4.0*
B =	5			*In function*
C =	2			*A,B,X,Y -*
D =	2			*Local variables*
X =	5			*C,D -*
Y =	5			*Global variables*
A =	6			*Step 4.25*
B =	5			*In function*
C =	3			*A,B,X,Y -*
D =	2			*Local variables*
R =	25			*C,D,R -*
X =	2.50000000			*Global variables*
Y =	5			
A =	2			*Step 1.4*
B =	2			*Outside function*
C =	3			*All global variables*
D =	2			
R =	25			
S =	30			
Z =	16.50000000			

```
END
>QUIT
```

NOTE: In the above example A and B are used as global variables outside the function but as local variables inside the function. X and Y exist only inside the function.

Recursive Functions

A recursive function is a programmer defined function that calls itself while evaluating an argument. Recursive functions are allowed in CAL and are demonstrated by the following example. This example uses a single statement, recursive programmer defined function to compute the factorial of a number.

```
>1 DEMAND X↵
>1.1 Y=F[X] ↵
>1.2 TYPE Y↵
>DEFINE F[X]=IF X>0↵
>THEN F[X-1]*X ELSE 1↵
>RUN↵
          X = 9↵
          Y =     362880
>
```

SECTION 7
LINES AND PAGES

The LINE, LINES, and PAGE commands put the user in control of the paging and spacing of the output.

LINE

The command LINE spaces the terminal paper up one line. It may be used both directly and indirectly with any of the CAL modifiers.

LINES

LINES is used with a positive integer to specify the number of lines that are to appear on a page. A maximum of 99 lines may be specified. CAL normally specifies 55 lines per page. This command may be used **directly** only and only when paging is specified. The paging may be automatic (automatic paging occurs in CAL when a heading is used) or may be defined in the program.

PAGE

The PAGE command spaces the terminal paper up one page. Unless otherwise specified, a page is assumed to be 55 lines. This command may be used both directly and indirectly with any of the CAL modifiers.

$

The $ stores the number of the current line. To determine the current line of the program, use the direct command TYPE $ and the current line number will be returned. The $ commonly is used with the IF modifier to space or page only under specified conditions.

Examples

```
> LINE IF ($MOD 5) = 0
```

The above statement will double space every five lines.

```
> PAGE IF $ = 40
> LINES 40
```

Both of these statements will set the number of lines per page at 40.

SECTION 8
PROGRAM FILES

A program written on the terminal in CAL may be saved and reused at any time by storing the program on a disk file.

To save a program, use the CAL DUMP command as follows:

> DUMP
TO /File Name/↵

NOTE: Any single character or group of characters may be used as a file name. It is suggested, however, that file names be short (1 - 4 characters).

Now CAL will respond with either NEW FILE or OLD FILE as follows:

NEW FILE

This message indicates that the file is a new one; that is, there is no file by that name. Press the Carriage Return if you want to create a new file.

OLD FILE ↵ or ⊕

This message indicates an attempt to write over (change) an old file. To change the old file, hit the Carriage Return to acknowledge that the file is an old one. If the file name is already in use, and should be saved, hit the ALT MODE/ESC and repeat the DUMP procedure using a new file name.

NOTE: Direct commands and values stored in variables will not be saved on disk files created with the CAL DUMP command.

To reuse a program which has been saved on the disk, use the LOAD command as follows: *NOTE: This command* **copies** *the file from the disk; the file is not erased.*

> LOAD ↵
FROM /File Name/ ↵
>

When the computer returns the >, the program is ready for use. Only statements with errors will be printed. To obtain a complete listing of the program use the command TYPE ALL when the > is returned. To start execution of the program, use the RUN command. To edit the program in any way, make the changes, and then use the DUMP command to write the edited program back on the file. Remember that nothing done on the terminal will be saved unless it is dumped back on the file.

Using A Tymshare Library Program

> LOAD ↵
FROM "Library File Name" ↵

Any library program which is written in CAL may be called by using the LOAD command and enclosing the file name in double quote marks as shown above. Most CAL library programs are self-starting because a direct TO PART or RUN is stored in the file.

Removing A File

To remove a previously saved file from the disk, use the EXECUTIVE command DELETE followed by the file name in slashes.

Data Files

A data file is exactly what the name implies; namely, a file on which numeric data is stored. Alphabetic data may be stored on a CAL data file but it will be ignored completely when the file is read since CAL reads only numeric data from data files. *CAUTION: CAL recognizes all periods (.) as decimal points. If a period is used in the alphabetic text, CAL automatically will assign a numeric value of 0.0 to the period.*

A data file used in CAL may have been created in CAL, EXECUTIVE, or EDITOR. Two types of data files may be created in CAL, Symbolic Files and Binary Files. Binary files generally require less storage space but have the disadvantage that the file may be read only by a CAL program. Symbolic files also may be created in EXECUTIVE or EDITOR. When using EXECUTIVE or EDITOR, a Carriage Return, a space, or a comma may be used to terminate the numbers.

Opening And Closing A Data File

All CAL data files, whether used for input or output, must be opened and closed. Data files are opened in the following manner:

OPEN /file name/ FOR $\begin{matrix}\text{INPUT}\\\text{OUTPUT}\end{matrix}$ AS FILE $\begin{matrix}\text{file}\\\text{number}\end{matrix}$

The file number is normally an integer, although a variable or an expression may be used.

To close a CAL data file, the CLOSE command is used with the number of the file as follows:

CLOSE file number ↵

CAL data files are closed automatically whenever the user returns to the EXECUTIVE. CONTINUE does not reopen data files.

After a file has been closed it may not be used again until it is reopened. The OPEN command always positions a file at the beginning of the data in that file. Once a file has been closed there is no way to continue reading or writing a file; it must be reread or rewritten from the beginning.

The OPEN and CLOSE commands may be used both directly and indirectly with any of the CAL modifiers except FOR, UNTIL, and WHILE.

NOTE: Only three files may be open at one time.

Data File Input

The READ and INPUT commands read numeric data from a disk file and store the data in the variables listed.

The READ command is used when the data to be read is stored in a symbolic data file. The INPUT command is used when a binary data file is read. (Binary data files are created with the CAL OUTPUT command.) The general format of the commands is as follows:

READ / INPUT FROM file number : variable list

Both of these commands may be used both directly and indirectly with any of the CAL modifiers.

The READ and INPUT commands recognize an end-of-file condition. If an attempt is made to read more data than exists on a file, CAL will issue an end-of-file error diagnostic and return control to the user. The programmer should know how many numbers will be on an input file at the time he writes the program OR he should establish some convention within the data on the file to tell the program when the end-of-data has been reached.

Two methods of indicating an end-of-file are:

1. The first number on a file could specify the number of numbers which are stored on the file.
2. The last number on the file could be some otherwise unused number. The last number would not be data for program computation, but rather a programmer defined end-of-data signal. A number such as 999.999 or 9.9999999 E-70 could be established as the special end-of-data signal.

Data File Output

To create a symbolic data file, CAL uses the command WRITE. The general format of the WRITE command is as follows:

WRITE ON file number : variables to be stored on the file

With a symbolic file, both the variables and their values are stored on the file. When the file is read back into CAL only the values are read (remember, CAL ignores non-numeric text). A problem arises if subscripted variables appear on a file. When the file is read, CAL will treat the subscripts as data. This problem is easily avoided by using a FORM statement with the WRITE command whenever subscripted variables are used. The general format is as follows:

WRITE ON file number IN FORM form number : variable list

FORM statements are discussed on Page 10. The output form specification is used. When a FORM statement is used, the programmer controls which information will be stored on the data file and the subscripted variables may easily be eliminated.

To create a binary file, CAL uses the command OUTPUT as follows:

OUTPUT ON file number : variable list

All of the variables specified in the variable list must be non-subscripted variables. In this way the problem encountered above with subscripted variables is avoided.

The above command may be used either directly or indirectly and may be used with any of the CAL modifiers.

Example

```
> 1.0 OPEN /R/ FOR OUTPUT AS FILE 1
> 1.1 A=1
> 1.2 B=2
> 1.3 C=A+B
> 1.4 D←E←7
> 1.5 WRITE ON 1: A,B,C,D,E
> 1.6 CLOSE 1
> RUN
> QUIT

– COPY /R/ TO TELETYPE
  A =          1
  B =          2
```

```
   C =          3
   D =          7
   E =          7

– CONTINUE
CAL

>2.0 OPEN /R/ FOR INPUT AS FILE 2
>2.2 READ FROM 2: R(I) FOR I=1 TO 5
>2.3 TYPE R
>2.4 CLOSE 2
>TO PART 2
   R(1) =       1
   R(2) =       2
   R(3) =       3
   R(4) =       7
   R(5) =       7
>QUIT
–
```

Example

```
>1.0 OPEN /T1/ FOR OUTPUT AS FILE X
>1.1 OPEN /T2/ FOR OUTPUT AS FILE X+1
>1.2 A(I)=I FOR I=1 TO 10
>1.3 WRITE ON X: A(I) FOR I=1 TO 10
>1.4 WRITE ON X+1 IN FORM 1: A(I) FOR I=1
                                    TO 10
>FORM 1:
%%%%%%%%
>1.5 CLOSE X
>1.6 CLOSE X+1

>X=1
>RUN
>QUIT

– COPY /T1/ TO TELETYPE
   A(1) =       1
   A(2) =       2
   A(3) =       3
   A(4) =       4
   A(5) =       5
   A(6) =       6
   A(7) =       7
   A(8) =       8
   A(9) =       9
   A(10)=      10

– COPY /T2/ TO TELETYPE
    1
    2
    3
    4
    5
    6
    7
    8
    9
   10
–
```

SECTION 9
CAL EDITING FEATURES

If you have made an error in your program, there are three courses of action open to you:

1. Add a statement.
2. Delete a statement.
3. Change a statement.

Adding A Statement

Adding a statement is easy in CAL. Just type the statement and CAL automatically inserts it into the program according to step number. For example, to add a statement between steps 1.0 and 1.1, simply type the statement using step number 1.05. Step numbers may range from 0.0 to 999.999.

Deleting A Statement

To delete a statement, several different commands may be used depending upon the type of statement to be deleted. All of the following commands must be executed **directly** (without step numbers).

DELETE STEP 1.0, 2.1, 3.5
DELETE 1.1

When used with a step number or a list of step numbers, the DELETE command deletes the steps specified (removes them from the program).

DELETE PART 1, N-1

DELETE PART, when used with a step or part number or an expression, deletes **all** of the statements in the specified part.

DELETE FORM 1

DELETE FORM, when used with a form number or an expression, deletes the form statement specified.

DELETE A

When used with a variable or a list of variables, the DELETE command deletes the values stored in the variables. When used with a function name, this command deletes the DEFINE statement in which the function was defined.

The DELETE commands may be used in plural form using the following commands:

DELETE ALL STEPS
DELETE ALL FORMS
DELETE ALL VALUES
DELETE ALL FUNCTIONS

DELETE ALL

This command deletes the entire program and asks for the number of statements again.

CLEAR

CLEAR works in the same manner as DELETE ALL. These two commands may be used interchangeably.

Changing A Statement

The easiest way to change a statement is to retype it using the same step number, form number, or, with a DEFINE statement, the same function name.

EDIT And MODIFY

A program also may be changed by using the appropriate EDIT or MODIFY (MOD) command presented below. These commands are used directly only (without a step number).

EDIT STEP 1.0
EDIT 1.0

These commands search the program for the statement whose step number has been specified, type out the entire statement, return control to the user with a >, and then wait for the user to change the statement. Use the control characters described below to change the statement.

EDIT FORM 1

This form of the EDIT command works in the same manner as the other EDIT commands to change form statements. The form number may be an integer or an expression.

EDIT A

The EDIT command is used with a function name to edit the DEFINE statement in which the function was defined.

MODIFY STEP 1.0
MODIFY 1.0
MODIFY FORM 1
MODIFY A

The MODIFY commands (which may be shortened to MOD) are used in the same manner as the EDIT commands. The only difference between EDIT and MOD is that the MOD commands do not type out the statement to be changed, but simply return control to the user with a > and wait for the user to change the statement.

NOTE: The EDIT and MODIFY (MOD) commands do not affect the statement being edited unless the same step number, form number, or programmer defined function name is used in the new line. For example, to change Step 1.5 to Step 2.0, not only the step number must be changed but also the original step (1.5) must be deleted.

Control Characters

The control characters explained below are used with the EDIT and MODIFY commands. They also may be used any time that a statement is being typed, and are thus useful in correcting errors at the time they occur. A control character is generated by pressing and holding the control key on the keyboard and then hitting the appropriate character. Control characters do not normally print, but if they do, a symbol and not the character pressed will be typed. Control characters are indicated by a superscript c; for example, A^c represents a Control A. A list of the control characters, the symbol printed, and the action generated is given below.

EDITING CONTROL CHARACTERS

Control Character	Symbol Printed	Action
For Deleting		
A^c	↑	Deletes the previous character. Repeated use deletes several characters.
Q^c	←	Deletes the entire line.
W^c	\	Deletes the preceding word in the line.
W^c (Data)	\	Deletes the data value being typed in response to a DEMAND command.
X^c *and a character*	%	Deletes up to and including the character typed after it.
S^c	%	Deletes the next character in the old line.
Carriage Return		Deletes the rest of the old lines and ends the edit.
For Inserting		
E^c	< >	Inserts text into the old line; first E^c prints <, second E^c prints >.
For Copying		
C^c		Copies the next character in the old line.
D^c		Copies and prints the rest of the old line and ends the edit.
F^c		Copies but does not print the rest of the old line and ends the edit.
R^c		Prints the rest of the old line plus the new line; edit continues.
T^c		Same as R^c except that it aligns old and new lines.
Y^c		Copies but does not print the rest of the old line; edit continues with the new line acting as old line.
Z^c *and a character*		Copies up to and including the character typed after it.

EDIT and MODIFY are quite useful and should be learned, as they will reduce considerably the time needed to type a program.

Example Using EDIT Command, DELETE Command, And Control Characters

```
>1.0 DEMAND A,B↵
>2.0 T=B↑2-SA^c↑A↵
```

```
> 2.0 S=RA^C↑T+A↑2Q^C←
> 2.1 S=T+A↑2
> 3.0 TYPE T,S
> 3.1 TYPE A,B
> TYPE ALL

1.0 DEMAND A,B

2.0 T=B↑2-A
2.1 S=T+A↑2

3.0 TYPE T,S
3.1 TYPE A,B
```

```
> EDIT 3.0
  3.0 TYPE T,S
> Z^CE3.0 TYPE A,B,T,S
> DELETE STEP 3.1
> TYPE ALL

1.0 DEMAND A,B

2.0 T=B↑2-A
2.1 S=T+A↑2

3.0 TYPE A,B,T,S

>
```

SECTION 10
LOGICAL OPERATIONS

In addition to the arithmetic operations already covered, CAL also is equipped to handle logical operations. Although logical operations appear to be similar to arithmetic operations, logical operations differ in that they recognize only two conditions; namely, true (not zero) and false (zero).

Logical Variables And Expressions

All arithmetic variables and expressions have a logical as well as an arithmetic value. Any arithmetic variable or expression whose numeric value is not zero is said to have a logical value of 1 (True). Any arithmetic variable or expression with a zero value is said to have a logical value of 0 (False).

Logical Comparisons

Logical comparisons are made using logical expressions or variables together with relational operators. Whenever two defined values are compared by means of relational operators, a logical operation is performed. A logical value of 1 is assigned if the relationship specified is true, and a logical value of 0 is assigned if the relationship is not true.

```
>TYPE 3=3
    3=3 =          1
>A = 3
>TYPE A>3+1
    A>3+1 =        0
```

Logical Operators

CAL has three logical operators (AND, OR, NOT) which work exclusively with logical values. CAL's logical operators may be used alone; for example, TYPE A AND B, or, as introduced earlier in this manual, in conjunction with the arithmetic and relational operators; for example,

TYPE A, A↑2 IF A↑2<33 AND B>A-5

AND

The logical operator AND evaluates the expressions immediately preceding and following it. If both of these expressions are true, a logical value of 1 (not zero) will be returned. If either one or both of the expressions is false, a logical value of false (0) is returned. AND works as a multiplication operator on the logical values, hence only the condition TRUE* TRUE (for example, 1*1) will produce a TRUE (not zero) value.

OR

The logical operator OR also evaluates the expressions immediately preceding and following it. OR, however, requires that only one of the logical expressions be true to return a true value. OR works as an addition operator on the logical value; thus a false value (0) will be returned only when both of the expressions are false (0). If either of the expressions connected by an OR is true, the result of the logical addition can never be 0 and hence will be TRUE (not zero).

NOT

The logical operator NOT changes the logical value of the expression immediately following it. Thus, if A is equal to 1 (TRUE), then NOT A is equal to 0 (FALSE). If B=0 (FALSE), then NOT B=1 (TRUE).

Example: Logical Operators

```
>LOAD
FROM /LOGIC/

>TYPE ALL

1.00 A0=NOT A
1.10 B0=NOT B
1.20 A1=A AND B
1.21 A2=A AND NOT B
```

```
1.22 A3=NOT A AND B
1.23 A4=NOT A AND NOT B
1.30 01=A OR B
1.31 02=A OR NOT B
1.32 03=NOT A OR B
1.33 04=NOT A OR NOT B
1.40 TYPE IN FORM 1: A,A1,01,B,A2,02,A0,A3,03,B0,A4,04
1.50 LINE

FORM 1:
A  = %          A AND B = %          A OR B = %
B  = %          A AND NOT B = %      A OR NOT B = %
NOT A  = %      NOT A AND B = %      NOT A OR B = %
NOT B  = %      NOT A AND NOT B = %  NOT A OR NOT B = %

> DO PART 1 FOR A=0,1 FOR B=0,1 ↵
A  = 0          A AND B = 0          A OR B = 0
B  = 0          A AND NOT B = 0      A OR NOT B = 1
NOT A  = 1      NOT A AND B = 0      NOT A OR B = 1
NOT B  = 1      NOT A AND NOT B = 1  NOT A OR NOT B = 2

A  = 1          A AND B = 0          A OR B = 1
B  = 0          A AND NOT B = 1      A OR NOT B = 2
NOT A  = 0      NOT A AND B = 0      NOT A OR B = 0
NOT B  = 1      NOT A AND NOT B = 0  NOT A OR NOT B = 1

A  = 0          A AND B = 0          A OR B = 1
B  = 1          A AND NOT B = 0      A OR NOT B = 0
NOT A  = 1      NOT A AND B = 1      NOT A OR B = 2
NOT B  = 0      NOT A AND NOT B = 0  NOT A OR NOT B = 1

A  = 1          A AND B = 1          A OR B = 2
B  = 1          A AND NOT B = 0      A OR NOT B = 1
NOT A  = 0      NOT A AND B = 0      NOT A OR B = 1
NOT B  = 0      NOT A AND NOT B = 0  NOT A OR NOT B = 0

>
```

SECTION 11
OVERLAY

Any CAL program which can be divided into sequentially executed parts or segments can be overlayed. The overlay feature of CAL is the ability to write a program larger than the computer memory available to the user by bringing parts of the program into memory from the disk at different times. Although the user can do this manually at his terminal, the overlay technique described here causes the computer to load and execute a number of separate files automatically until the entire program is run. Thus the user has at his disposal a computer whose memory capacity is infinite (at least theoretically), since the only limit to the number of files a user may create is the size and number of files in his file directory.

To use the overlay feature of CAL, first write the program in CAL and then use the EDITOR commands READ, WRITE, and APPEND. Proceed as follows:

– CAL ↵ > >	*Call CAL and write the first segment or overlay of the program.*
> DUMP ↵ TO/FIRST/ ↵ NEW FILE ↵	*Write this section on a file with the CAL command DUMP. Use any file name which has not been used before.*
> QUIT ↵	*Remember that direct commands will not be saved.*
– EDITOR ↵ * READ/FIRST/ ↵	*Call EDITOR and read the file with the READ command.*
* APPEND ↵	*Use the APPEND command to add the direct commands needed.*
RUN ↵	*The RUN command will start execution of the program automatically.*
DELETE ALL STEPS ↵ DELETE FORM 1 ↵	*DELETE any steps, forms, functions, and values that will not be needed in later segments of the program. NOTE: Be sure to delete each part separately. DO NOT use the DELETE ALL or CLEAR commands because these commands will interrupt the overlay procedure by returning control to the user.*
LOAD ↵ /SECOND/ ↵	*Load the next file in the program.*
D^C	*Type a Control D to terminate the APPEND command.*
*WRITE/FIRST/ ↵ OLD FILE ↵ *QUIT ↵ –	*Use the WRITE command to place the edited version of the original file back on the disk.*

Repeat the above procedure for each overlayed program segment required.

– CAL ↵	*Now call CAL and begin the entire program by calling the first section of the overlay.*
LOAD ↵ FROM/FIRST/ ↵	

Overlay Example

Shown below is an example of an overlayed program. Note that the second file calls the first file which puts the program into a loop. Get out of the loop in the usual manner by hitting the ALT MODE/ESC key.

```
– CAL ↵
STATEMENTS = ↵

> 1.0 TYPE "THIS IS THE FIRST FILE OF THE PROGRAM." ↵
> 2.0 DEMAND A ↵
> 3.0 X=A↑2 ↵
> 4.0 TYPE IN FORM 1: A,X ↵
```

```
> FORM 1: ↵
              THE SQUARE OF  %%%%%%.%%%  IS  ############### ↵
> DUMP ↵
TO /FIRST/ ↵
NEW FILE ↵
> QUIT ↵
– EDITOR ↵
* READ /FIRST/ ↵

* APPEND ↵
RUN ↵
DELETE STEP 2.0 ↵
DELETE FORM 1 ↵
LOAD ↵
/SECOND/ ↵
* WRITE /FIRST/ ↵
OLD FILE ↵
76 WORDS.
* QUIT ↵

– CAL ↵
STATEMENTS = ↵

> 1.0 TYPE "THIS IS THE SECOND FILE OF THE PROGRAM." ↵
> 3.0 X=A↑3 ↵
> FORM 1: ↵
              THE CUBE OF  %%%%%%.%%%  IS  ############### ↵
> DUMP ↵
TO /SECOND/ ↵
NEW FILE ↵

> QUIT ↵
– EDITOR ↵

* READ /SECOND/ ↵

* APPEND ↵
RUN ↵
DELETE STEP 1.0 ↵
DELETE FORM 1 ↵
DELETE A ↵
LOAD ↵
/FIRST/ ↵
* WRITE /SECOND/ ↵
OLD FILE ↵
65 WORDS.
* QUIT ↵
– CAL ↵
STATEMENTS = ↵
```

```
>LOAD ↵
FROM /FIRST/ ↵

THIS IS THE FIRST FILE OF THE PROGRAM.
        A  = 5.0 ↵
                THE SQUARE OF        5.000     IS      2.5000000E 01
FROM
THIS IS THE SECOND FILE OF THE PROGRAM.
        A  = 5 ↵
                THE CUBE OF          5.000     IS      1.2500000E 02
FROM
THIS IS THE FIRST FILE OF THE PROGRAM.
        A  = 32.5 ↵
                THE SQUARE OF        32.500    is      1.0562500E 03
FROM ⊕
```

SECTION 12
COMMAND FILES

Issuing Commands From A File

The EXEC command

```
-COMM /file name/↵
```

instructs the system to take its commands from a file instead of from the terminal. The user simply creates a file containing all the commands which he wants executed. The commands may be from any language, including EXEC, and are typed into the file exactly as they would normally be given from the terminal.

Creating A Command File

A command file may be created in the EXEC (with the COPY TEL TO /file name/↵ command) as follows. This command file is created to call CAL, indicate 97 statements, no heading, and then load and run a CAL program.

```
-COPY TEL TO /C4/↵
  NEW FILE↵          Equivalent Terminal Commands:
CAL↵                 -CAL↵
97↵                  STATEMENTS=97↵
↵                    HEADING=↵
LOAD↵                >LOAD↵
/FACTOR/↵            FROM /FACTOR/↵
RUN↵                 >RUN↵
QUIT↵                >QUIT↵
COMM TEL↵            -COMM TEL↵
D^C
-
```

Leaving A Command File

The system will take its commands from the file specified in the COMM command until one of the following is reached:

1) The end of the command file, which causes the system to return to taking commands from the terminal.
2) A COMM TEL↵ command, which has the same effect as 1.
3) Another COMM command that enables the user to nest command files as deeply as he wishes. *NOTE: Command files can be recursive; that is, the last command in the file can be a command to take commands from itself.*

The CAL program loaded in the preceding example is on a file called /FACTOR/ which was created as follows:

```
-CAL↵
STATEMENTS =↵
>1 F=1↵
>1.1 DEMAND A↵
>2 F=F*N FOR N=1 TO A↵
>3 TYPE F↵
>DUMP↵
TO /FACTOR/↵
 NEW FILE↵

>
```

To tell EXEC to start taking commands from /C4/, type the EXEC command:

```
-COMM /C4/↵
```

The output on the terminal which results from an actual run of this example is:

```
-COMM /C4/↵
STATEMENTS =
HEADING =
>FROM
>          A = 11↵
         F =  39916800

-
```

Instead of having two files as shown above to execute the program, a single file can be used both as a source of commands and for the program itself as shown below.

```
-COPY TEL TO /CC/↵   The command file CC is
  NEW FILE↵          created.

CAL↵
97↵
↵
1 F=1↵
1.1 DEMAND A↵
2 F=F*N FOR N=1 TO A↵
3 TYPE F↵
RUN↵
QUIT↵
COMM TEL↵
D^C

-COMM /CC/↵          Now, /CC/ is executed.
STATEMENTS =
HEADING =
>          A = 9↵
         F =    362880

-
```

SECTION 13
SAMPLE PROGRAMS

This section contains sample programs and executions designed to demonstrate the features of CAL. Each program is in the following format:

1. **Define The Problem**

   The problem is explained and written in simple steps which are set in the following form:

   A. **Input.** All data which must be supplied by the user.

   B. **Compute.** All computations done by the computer.

   C. **Output.** All data which will be returned to the user.

2. **Flowchart**

3. **CAL Code and Sample Execution**

   The problem is coded in CAL and then run on the computer. Many of the examples in this section were reproduced from terminal hard copy.

The sample programs are given in order of difficulty. The first programs are simple, using only the basic CAL commands. The programs become progressively longer and more complex. Each program demonstrates a different set of CAL commands.

MONTHLY PAYMENT PROGRAM

1. Define The Problem

The problem is to compute the monthly payment on a debt.

A. **Input:**

1. Description - YES or NO.
2. Original debt (P)
3. Annual interest (I)
4. Number of monthly payments to be made (N)

B. **Compute:**

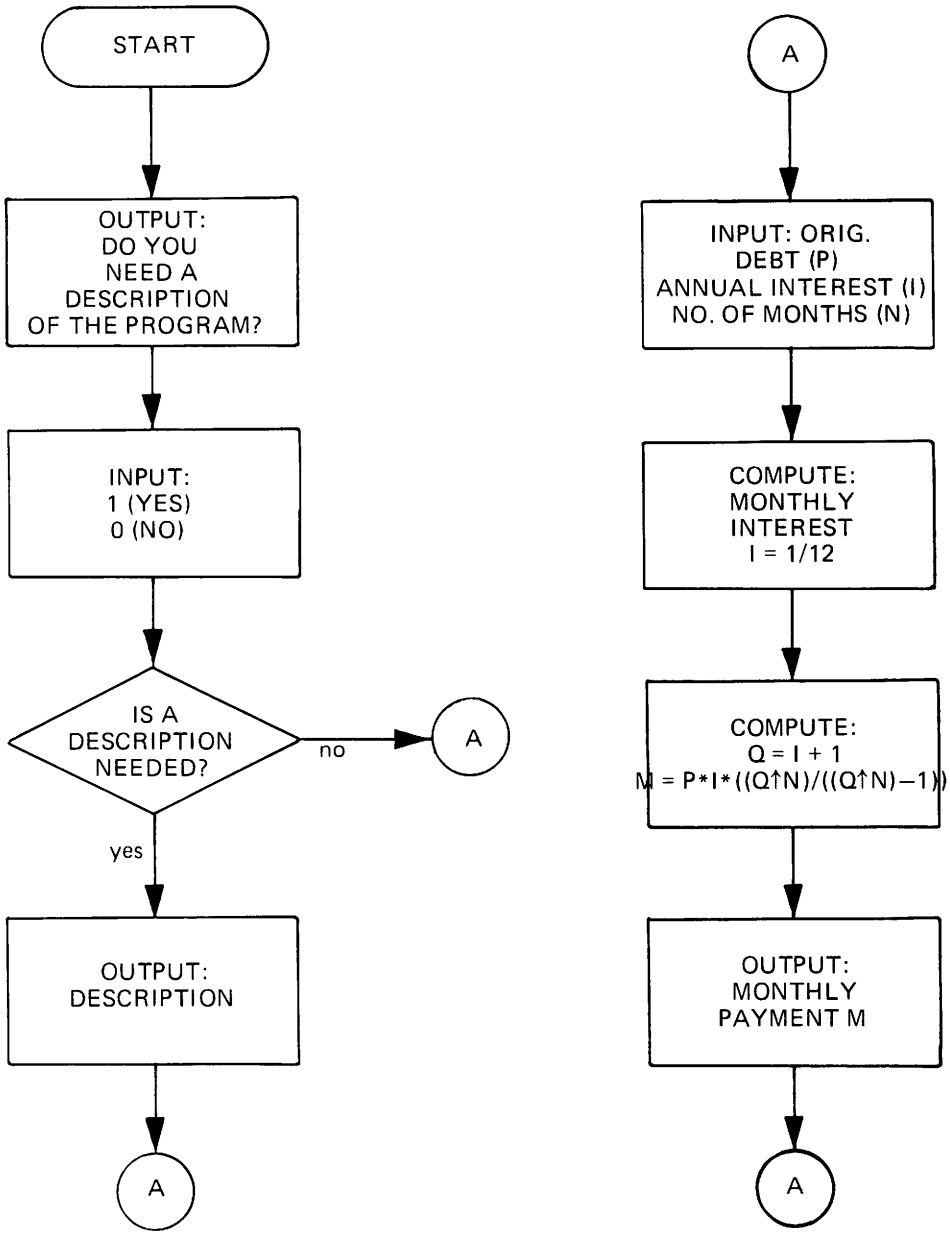
Monthly payment (M)

$$M = \frac{P \cdot I(I+1)^N}{(I+1)^N - 1}$$

C. **Output:**

Monthly payment, M.

2. Flowchart


START
OUTPUT:
DO YOU
NEED A
DESCRIPTION
OF THE PROGRAM?
INPUT:
1 (YES)
0 (NO)
IS A
DESCRIPTION
NEEDED?
no
A
yes
OUTPUT:
DESCRIPTION
A
A
INPUT: ORIG.
DEBT (P)
ANNUAL INTEREST (I)
NO. OF MONTHS (N)
COMPUTE:
MONTHLY
INTEREST
I = 1/12
COMPUTE:
Q = I + 1
M = P*I*((Q↑N)/((Q↑N)−1))
OUTPUT:
MONTHLY
PAYMENT M
A

3. CAL Code And Sample Execution

```
>LOAD
FROM /MOPAY/
```

This program was loaded from the file /MOPAY/ which had been created earlier.

```
>TYPE ALL

0.0 TYPE "TYPE '1' TO GET DESCRIPTION."
0.1 TYPE "TYPE '0' TO SKIP DESCRIPTION."
0.2 DEMAND IN FORM 1:D
0.3 TO PART 1 IF D#1
0.4 TYPE "THIS PROGRAM REQUESTS THE USER TO SUPPLY THE FOLLOWING"
0.5 TYPE "INFORMATION ON A LOAN - ORIGINAL DEBT(P), ANNUAL INTEREST(I)"
0.6 TYPE "AND NUMBER OF MONTHS(N).  IT COMPUTES THE MONTHLY PAYMENT(M)"
0.7 TYPE "TYPES OUT THE MONTHLY PAYMENT AND ASKS FOR NEW LOAN DATA."

1.0 DEMAND IN FORM 2: P,I,N
1.1 I=I/12
1.3 M=P*I*(I+1)↑N/((I+1)↑N-1)
1.4 TYPE IN FORM 3: M
1.5 TO PART 1

FORM 1:
#

FORM 2:
PRINCIPAL = #  INTEREST = #  NO. OF MONTHS = #
FORM 3:
MONTHLY PAYMENT = $%%%%%%.%%

>RUN
TYPE '1' TO GET DESCRIPTION.
TYPE '0' TO SKIP DESCRIPTION.
0

PRINCIPAL = 1000
  INTEREST = .06
  NO. OF MONTHS = 24

MONTHLY PAYMENT = $     44.32

PRINCIPAL = 21100    INTEREST = .055    NO. OF MONTHS = 45

MONTHLY PAYMENT = $    519.97

PRINCIPAL =
>
```

ARITHMETIC MEAN OF A SERIES OF NUMBERS

1. Define The Problem

The problem here is to determine the mean of any series of numbers.

A. **Input:**

A series of numbers terminated by 1×10^{70}

B. **Compute:**

The mean (average) of the numbers

$$\text{MEAN} = \frac{\text{sum of the numbers}}{\text{number of numbers}}$$

C. **Output:**

The Mean, M.

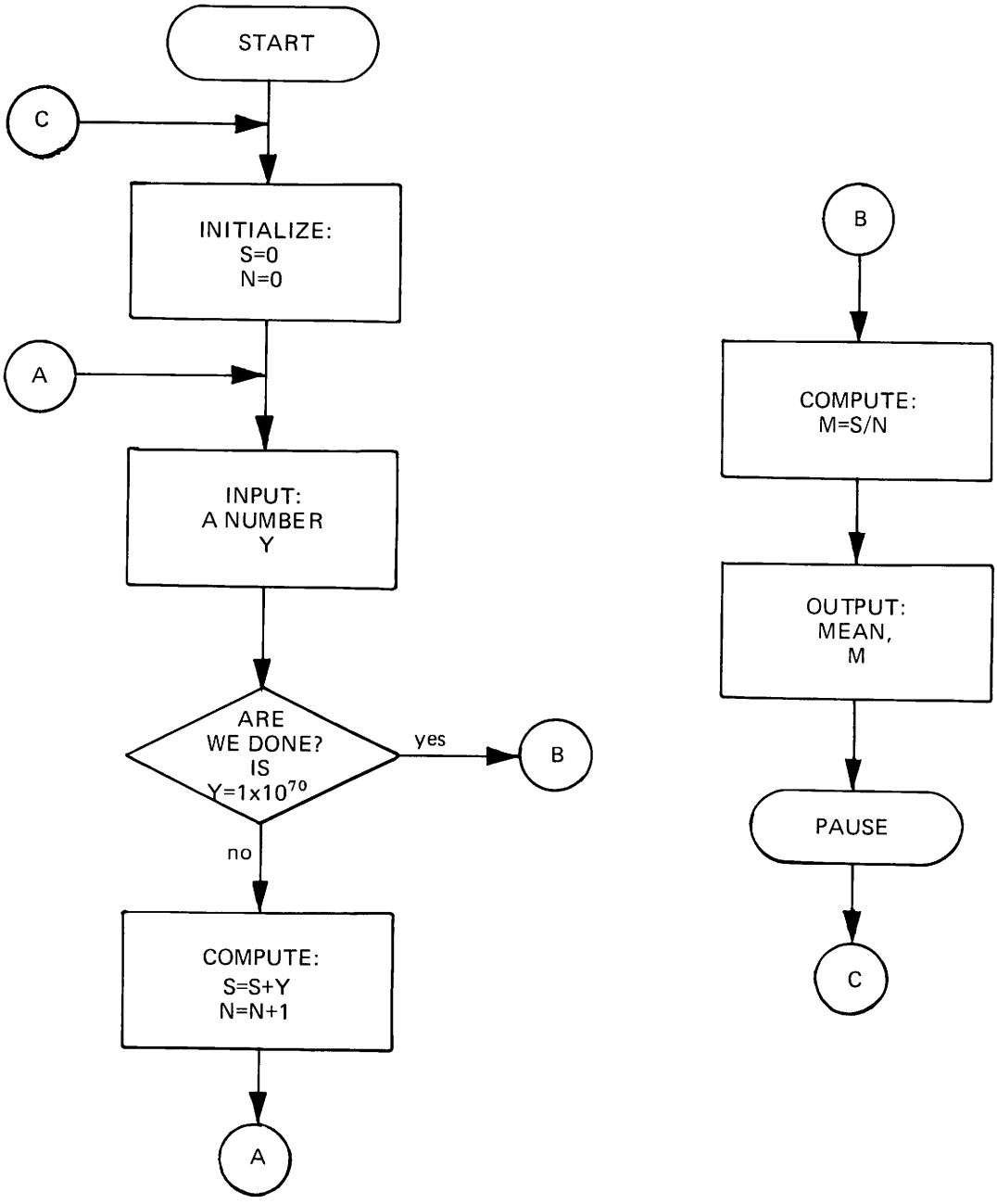
3. CAL Code And Sample Execution

```
> 1.0 S=N=0
> 1.1 DEMAND Y
> 1.2 TO PART 3 IF Y=1E70
> 1.4 S=S+Y
> 1.5 N=N+1
> 1.6 TO STEP 1.1
>
> 3.0 M=S/N
> 3.1 TYPE M
> 3.2 PAUSE
>
> 4.0 TO PART 1

>RUN
        Y = 24.3          Y = 66.7          Y = 109.6          Y = 3.0
        Y = 33.8          Y = 55          Y = 44.3          Y = 345
        Y = 65.4          Y = 1E70
        M =          83.01111100

PAUSE IN STEP 3.2:
>
```

2. Flowchart


START
C
INITIALIZE:
S=0
N=0
A
INPUT:
A NUMBER
Y
ARE
WE DONE?
IS
Y=1x10^70
yes
B
no
COMPUTE:
S=S+Y
N=N+1
A
B
COMPUTE:
M=S/N
OUTPUT:
MEAN,
M
PAUSE
C

AUTOMOBILE GAS MILEAGE

1. Define The Problem

The problem below computes the total mileage, the gas mileage, and the cost per mile for any given number of miles, given the initial odometer reading, the number of gallons used and the cost for each tankful, and the final odometer reading.

A. **Input:**

1. Initial odometer reading (I).
2. Number of gallons of gas per tankful (G) terminated by 0.
3. Cost per tankful (C) terminated by 0.
4. Final odometer reading (F).

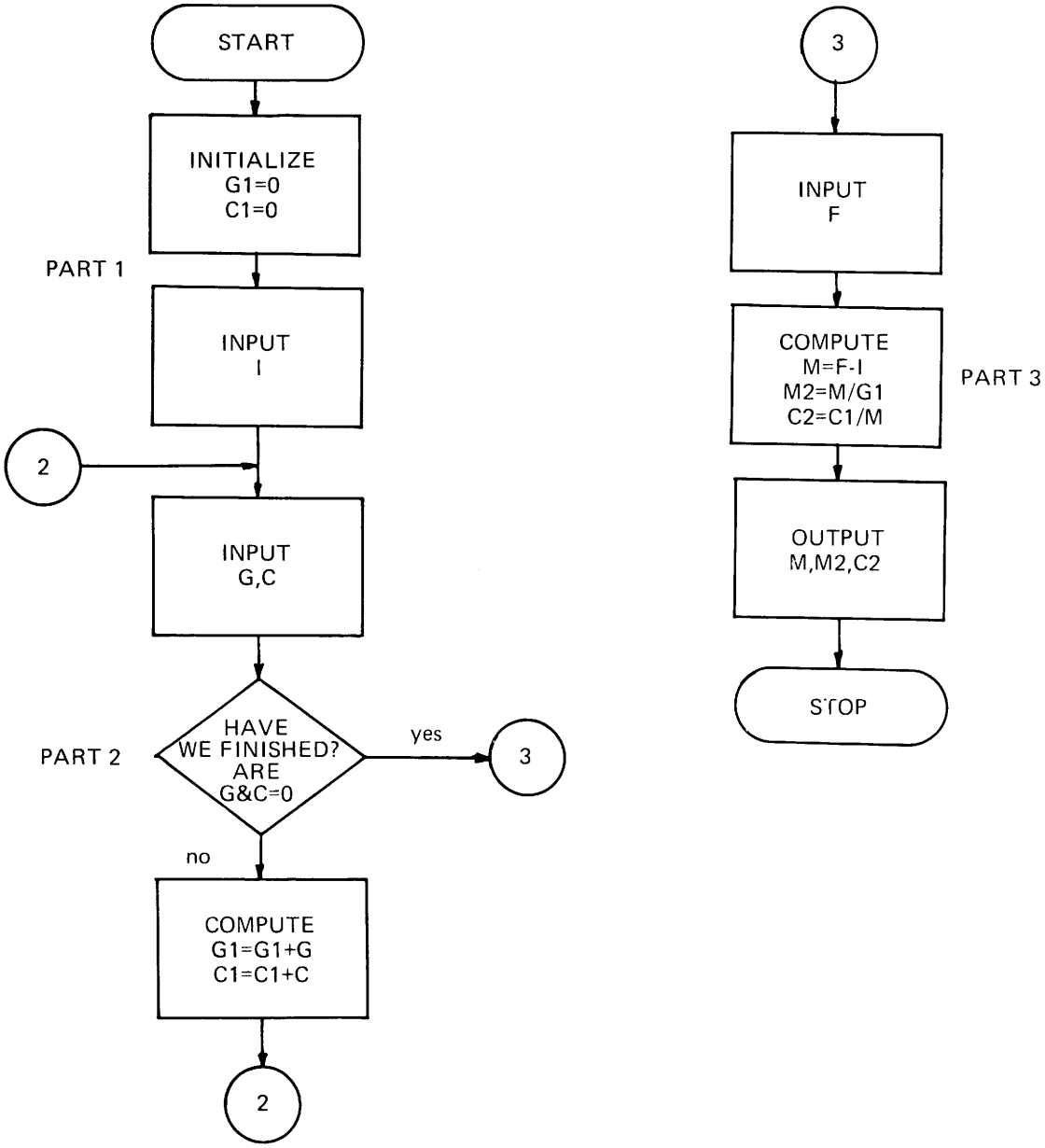
B. **Compute:**

1. Total miles travelled.
2. The number of miles/gallon for the trip.
3. The cost/mile for the trip.

C. **Output:**

1. Total miles travelled. (M)
2. Miles/gallon. (M2)
3. Cost/mile. (C2)

2. Flowchart


START
INITIALIZE
G1=0
C1=0
PART 1
INPUT
I
2
INPUT
G,C
PART 2
HAVE
WE FINISHED?
ARE
G&C=0
yes
3
no
COMPUTE
G1=G1+G
C1=C1+C
2
3
INPUT
F
COMPUTE
M=F-I
M2=M/G1
C2=C1/M
PART 3
OUTPUT
M,M2,C2
STOP

3. CAL Code And Sample Execution

```
-CAL
STATEMENTS =

>LOAD
FROM /MILE/

>TYPE ALL
```

This program was loaded from the file /MILE/ which had been created earlier.

```
1.0 TYPE "THIS PROGRAM COMPUTES THE GAS MILEAGE, THE COST/GAL,"
1.1 TYPE "AND THE TOTAL NUMBER OF MILES TRAVELED FOR ANY GIVEN"
1.2 TYPE "NUMBER OF MILES AND NUMBER OF FILLUPS."
1.5 G1=0
1.6 C1=0
1.8 DEMAND IN FORM 1: I
1.9 TYPE "    NUMBER OF GAL. PER FILLUP          COST PER FILLUP
          GAL =                              COST =    "

2.0 DEMAND IN FORM 2: G,C
2.1 TO PART 3 IF G=0 AND C=0
2.2 G1=G1+G
2.3 C1=C1+C
2.4 TO PART 2

3.0 DEMAND IN FORM 3: F
3.1 M=F-I
3.2 M2=M/G1
3.3 C2=C1/M
3.4 TYPE IN FORM 4: M,M2,C2
3.5 PAUSE
3.6 TO PART 1.5

FORM 1:
       INITIAL ODOMETER READING = #
FORM 2:
          # GAL                          $  #
FORM 3:
       FINAL ODOMETER READING = #
FORM 4:
          %%%%%%%%%%.%  MILES TOTAL
             %%%.%%     MILES/GAL        $  %%%.%% COST/MILE
```

```
>TO PART 1
THIS PROGRAM COMPUTES THE GAS MILEAGE, THE COST/GAL,
AND THE TOTAL NUMBER OF MILES TRAVELED FOR ANY GIVEN
NUMBER OF MILES AND NUMBER OF FILLUPS.
       INITIAL ODOMETER READING = 539

    NUMBER OF GAL. PER FILLUP          COST PER FILLUP
          GAL =                            COST =
          8.5  GAL                         $  3.21

          24.0  GAL                        $  8.60

          10.6  GAL                        $  4.01

          18.0  GAL                        $  6.10

          22.4  GAL                        $  8.50

            0  GAL                        $  0

       FINAL ODOMETER READING = 1494

             955.0  MILES TOTAL
           11.44    MILES/GAL        $    0.03 COST/MILE

PAUSE IN STEP 3.5:
>
```

DOUBLE DECLINING BALANCE DEPRECIATION

1. Define The Problem

The problem is to compute the double declining balance depreciation on any given asset over any specified number of years.

A. **Input:**

1. Cost of the asset (C).
2. Estimated useful lifetime (L).

B. **Compute:**

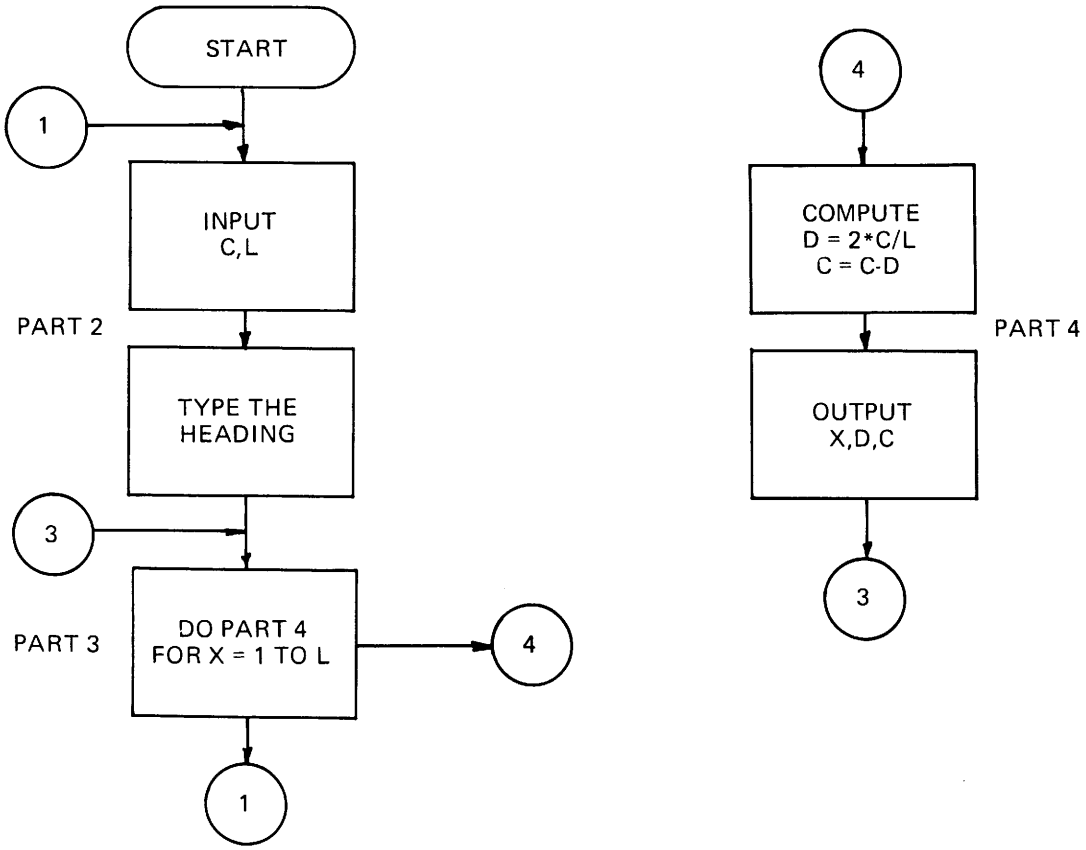
1. Depreciation $D = 2 \frac{C}{L}$
2. Book values $C = C - D$

C. **Output:**

For the entire range of years.

1. Year (X)
2. Amount of depreciation (D)
3. Book value (C)

2. Flowchart



3. CAL Code And Sample Execution

```
>LOAD
FROM /DEPR/
```

NOTE: Direct RUN stored on program using EDITOR.

```
PROGRAM TO CALCULATE DOUBLE DECLINING BALANCE DEPRECIATION


COST OF ASSET IS $100000
ESTIMATED USEFUL LIFE(IN YEARS) IS 20


YEAR    DEPRECIATION        BOOK VALUE

   1    $  10000.00         $  90000.00
   2    $   9000.00         $  81000.00
   3    $   8100.00         $  72900.00
   4    $   7290.00         $  65610.00
   5    $   6561.00         $  59049.00
   6    $   5904.90         $  53144.10
   7    $   5314.41         $  47829.69
   8    $   4782.97         $  43046.72
   9    $   4304.67         $  38742.05
  10    $   3874.20         $  34867.84
  11    $   3486.78         $  31381.06
  12    $   3138.11         $  28242.95
  13    $   2824.30         $  25418.66
  14    $   2541.87         $  22876.79
  15    $   2287.68         $  20589.11
  16    $   2058.91         $  18530.20
  17    $   1853.02         $  16677.18
  18    $   1667.72         $  15009.46
  19    $   1500.95         $  13508.52
  20    $   1350.85         $  12157.67


COST OF ASSET IS $
>TYPE ALL
```

```
0.0 LINE FOR I = 1 TO 5
0.1 TYPE "PROGRAM TO CALCULATE DOUBLE DECLINING BALANCE DEPRECIATION"
0.2 LINE FOR I = 1 TO 3
0.3 DEMAND IN FORM 1: C,L
0.4 LINE
0.5 TYPE "
YEAR   DEPRECIATION      BOOK VALUE"
0.6 LINE
0.7 DO PART 1 FOR X = 1 TO L
0.8 LINE FOR I = 1 TO 10
0.9 TO STEP 0.3

1.0 D = 2*C/L
1.1 C = C-D
1.3 TYPE IN FORM 2: X,D,C

FORM 1:
COST OF ASSET IS $#ESTIMATED USEFUL LIFE(IN YEARS) IS #
FORM 2:
 %%%   $%%%%%%%.%%      $%%%%%%%.%%

       C =     12157.66500000
       D =      1350.85170000
       I =        11
       L =        20
       X =        21

>
```

MEAN AND STANDARD DEVIATION

1. Define The Problem

The problem here is to compute the mean and standard deviation of a group of data. The mean is computed using the following formula:

$$M = \frac{\sum_{i=1}^{N} X_i}{N}$$

The standard deviation is computed using the following formula:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (X_i - M)^2}{N-1}}$$

A. **Input:**

1. The total number of data items (N).
2. The data (placed in the array A(I)).

B. **Compute:**

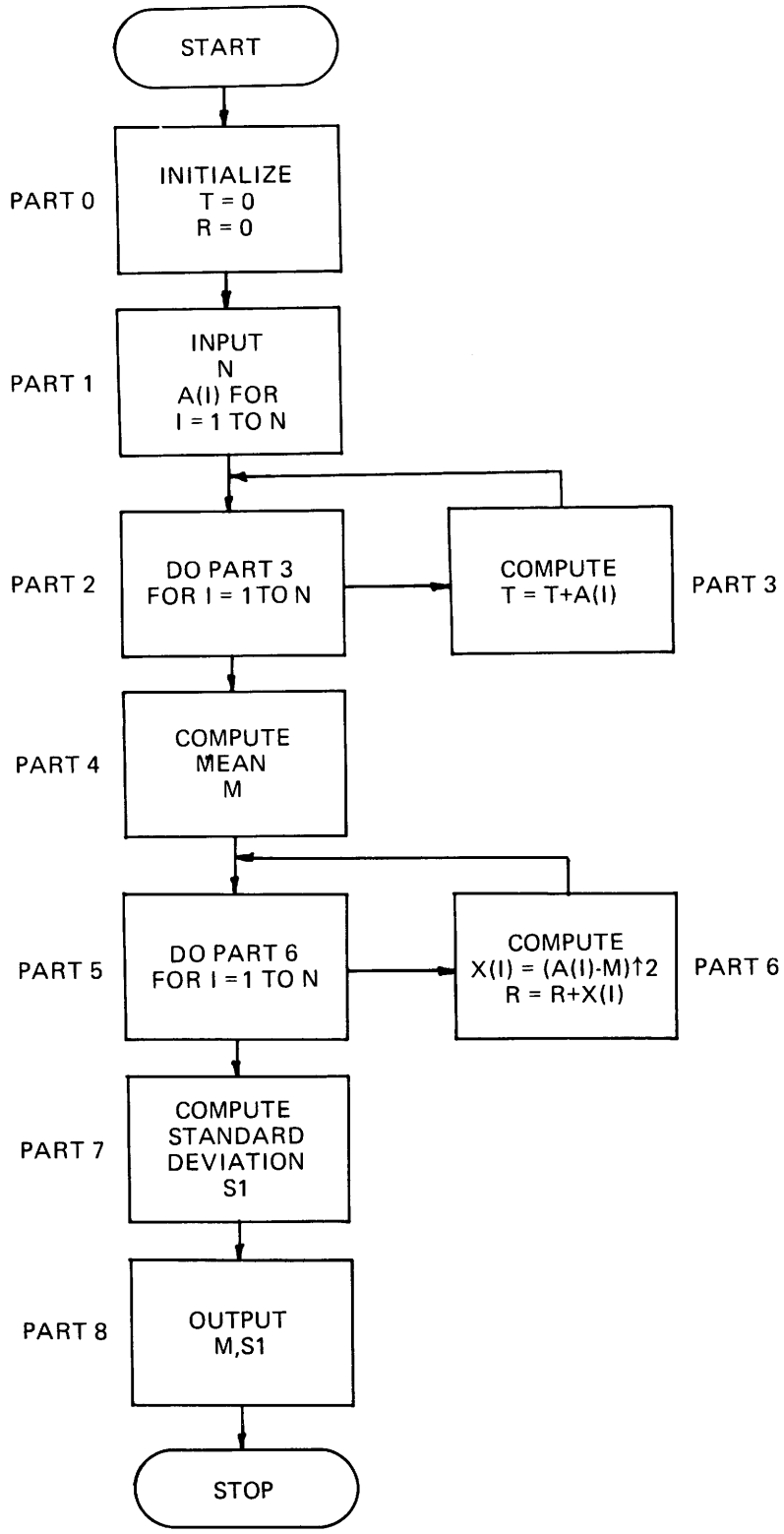
1. Mean
2. Standard deviation

C. **Output:**

1. Mean (M)
2. Standard deviation (S1)

METHOD 1

2. Flowchart

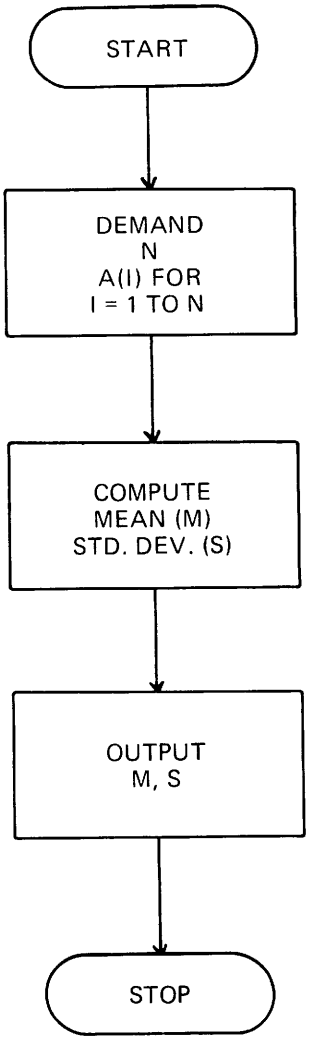


3. CAL Code And Sample Execution

```
>0.1 R←T←0
>1.0 DEMAND N
>1.1 DEMAND A(I) FOR I=1 TO N
>2.0 DO PART 3 FOR I=1 TO N
>2.1 TO PART 4
>3.0 T=T+A(I)
>4.0 M=T/N
>5.0 DO PART 6 FOR I=1 TO N
>5.1 TO STEP 7.0
>6.0 X(I)=(A(I)-M)↑2
>6.1 R=R+X(I)
>7.0 S1=SQRT(R/(N-1))
>8.0 TYPE M,S1
>RUN
        N = 5
   A(1)  = 2    A(2)  = 3    A(3)  = 1    A(4)  = 4    A(5)  = 2
        M =          2.40000000
       S1 =          1.14017540
>
```

METHOD 2

2. Flowchart



M = (I = 1 TO N:A(I))/N
S = SQRT (SUM(I = 1 TO N:(A(I) - M)↑2)/(N - 1))

3. CAL Code And Sample Execution

```
>1.0 DEMAND N
>1.1 DEMAND A(I) FOR I=1 TO N
>2.0 M=SUM(I=1 TO N:A(I))/N
>3.0 S=SQRT(SUM(I=1 TO N:(A(I)-M)↑2)/(N-1))
>4.0 TYPE M,S
>RUN
        N = 5
   A(1)  = 2     A(2)  = 3     A(3)  = 1     A(4)  = 4     A(5)  = 2
        M =            2.40000000
        S =            1.14017540
>
```

HISTOGRAM

A histogram or bar chart is a pictorial graph. The following example demonstrates how a histogram may be generated in CAL. The data is supplied from a data file.

DATA FILE FORMAT

The data ranges in value from 1 to 9. A programmer defined end-of-data signal (1E40) is used. The data file was created in the EXEC with the command COPY TEL TO /file name/.

1. Define The Problem

A. **Input:**

1. The scaling factor (S)
2. READ the data from the file one piece at a time and store it in the array (N).

B. **Compute:**

1. Increment the data counter (M)
2. Check to see if all of the data has been input (compare N with end-of-data signal 1E40)
3. Scale the data N=N/S
4. Increment the proper value of K(N)
5. The maximum value of N for which K(N) #0

C. **Output:**

The histogram of the data. The histogram is output using nested DO PART's. Study the example.

2. Flowchart

This problem can be coded directly without using a flowchart.

3. CAL Code And Sample Execution

```
-COPY TEL TO /DATA/
 NEW FILE

1,2,3,4,3,2,1,3,3,4,4,5,6,7,6,5,4,3,4,5,6,7,8,9,9,8,7,6,5,5,4,3,2
1,2,3,2,,1,2,3,4,5,6,7,8,9,8,7,6,5,4,3,4,5,6,7,8,,1,2,2,1,2,2,1
1,2,3,3,2,3,3,4,5,4,3,4,5,5,6,7,6,5,4,3,2,3,4,5,6,7,8,9,8,9,8,7,6
1E60

-CAL

STATEMENTS = 20
HEADING = HISTOGRAM



     PAGE 1

HISTOGRAM

>1.5 DEMAND IN FORM 1:S
>1.7 OPEN /DATA/ FOR INPUT AS FILE 1
>1.8 K(N)=0 FOR N=0 TO 500
>1.9 M=0
>2.5  READ FROM 1:N
> 2.53 M=M+1
>2.54 TO PART 10 IF N>=1E40
>2.55 N=N/S
>2.6 K(N)=K(N)+1
>2.7 TO STEP 2.5
>10.4 N=500
>10.5  N=N-1 UNTIL K(N)#0
>10.6 DO PART 50 FOR R=1 TO N
>10.8 LINE
>10.81 LINE
>10.9 TO PART 999999
>50.5 TYPE IN FORM 5:R/S
>50.6 DO PART 60 FOR A=1 TO K(R)
>50.7 LINE
>60.5 TYPE IN FORM 7:
>999999 LINE
>
>FORM 1:
SCALE FACTOR =#
>FORM 5:
%%%%.%%% :#
>FORM 7:
X#

>TO PART 1
SCALE FACTOR =1
```

NOTE: The extra field specified in the form statements (#) is used to suppress the Carriage Return. Thus an X is typed on the same line each time the loop is repeated. FOR A = 1 TO K(R).

```
    1.000 :XXXXXXXX
    2.000 :XXXXXXXXXXXXX
    3.000 :XXXXXXXXXXXXXXXX
    4.000 :XXXXXXXXXXXXXX
    5.000 :XXXXXXXXXXXXX
    6.000 :XXXXXXXXXXX
    7.000 :XXXXXXXXX
    8.000 :XXXXXXXX
    9.000 :XXXXX


>TYPE M
       M =          98
>QUIT
```

STANDARD MORTGAGE

1. Define The Problem

The problem is to compute the down payment, monthly payment, interest paid, equity accumulated, and the new balance for a standard mortgage. Three different types of mortgages may be requested; namely, F.H.A., commercial, and conventional.

A. **Input:**

1. Answers to Program Requests:
   a. Do you need instructions? YES or NO.
   b. Number of months remaining in first year =
   c. Original debt =
   d. Interest rate
   e. Type of mortgage: 1 = FHA; 2 = Conventional S & L; 3 = Commercial.
   f. The year in which mortgage payments begin =
   g. "Do you want to run another set of data?"

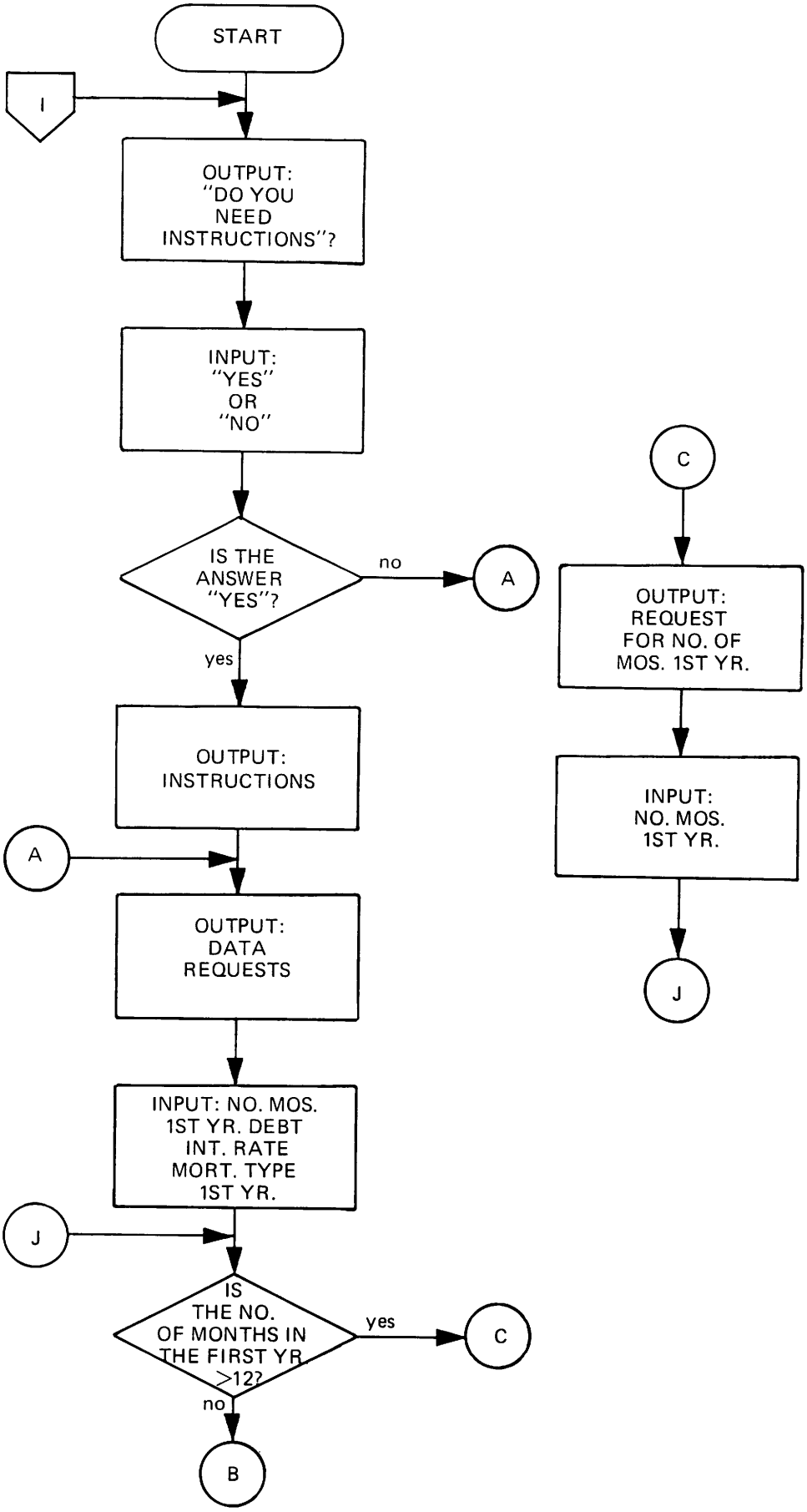
B. **Compute:**

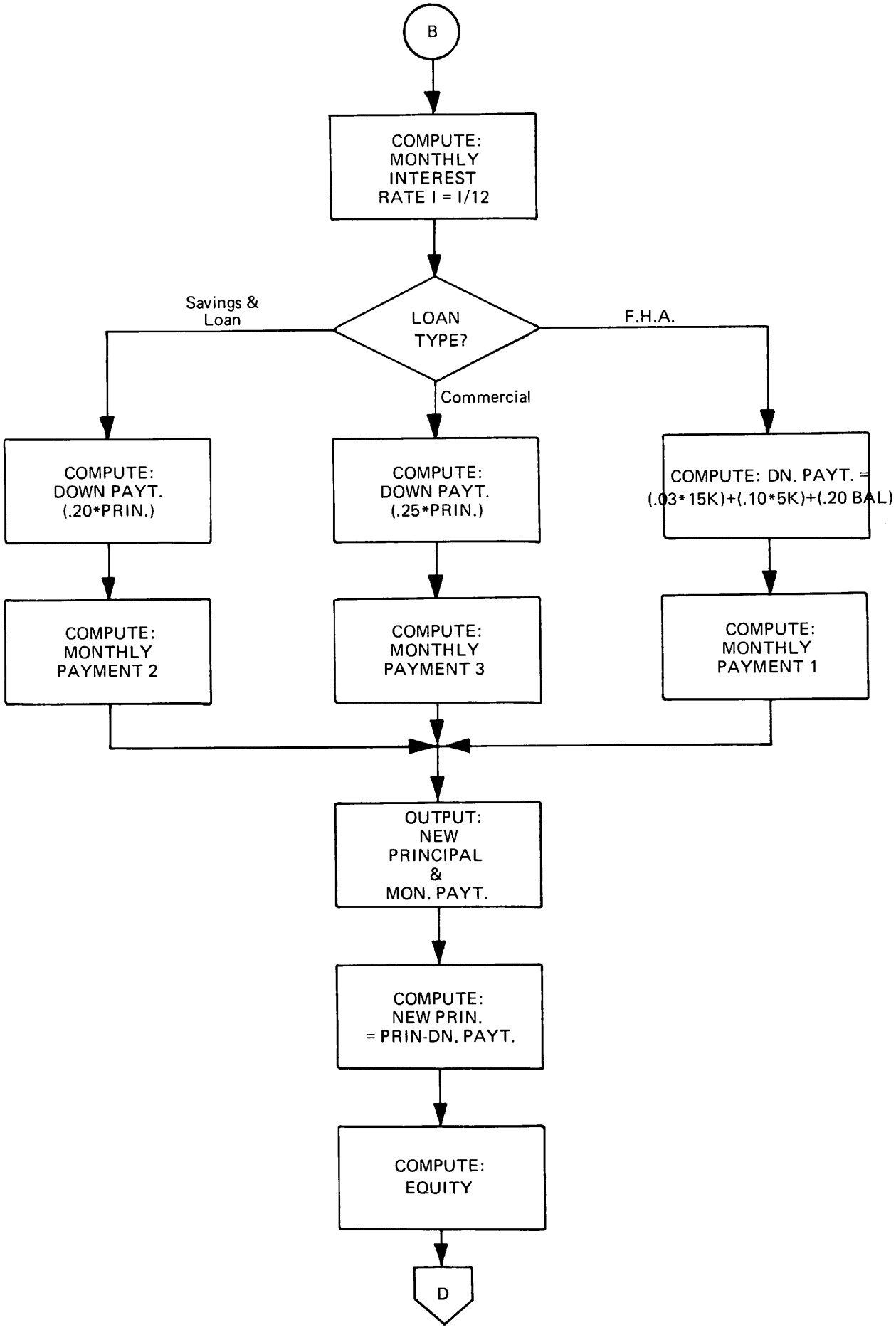
1. Minimum down payment
2. Monthly interest rate
3. New principal
4. Initial equity
5. Monthly payment
6. Number of months (based on mortgage type)
7. Initialize counters
8. Interest, equity, title interest, title equity, and principal for each month.
9. Interest, equity, title interest, title equity, and principal for each year.

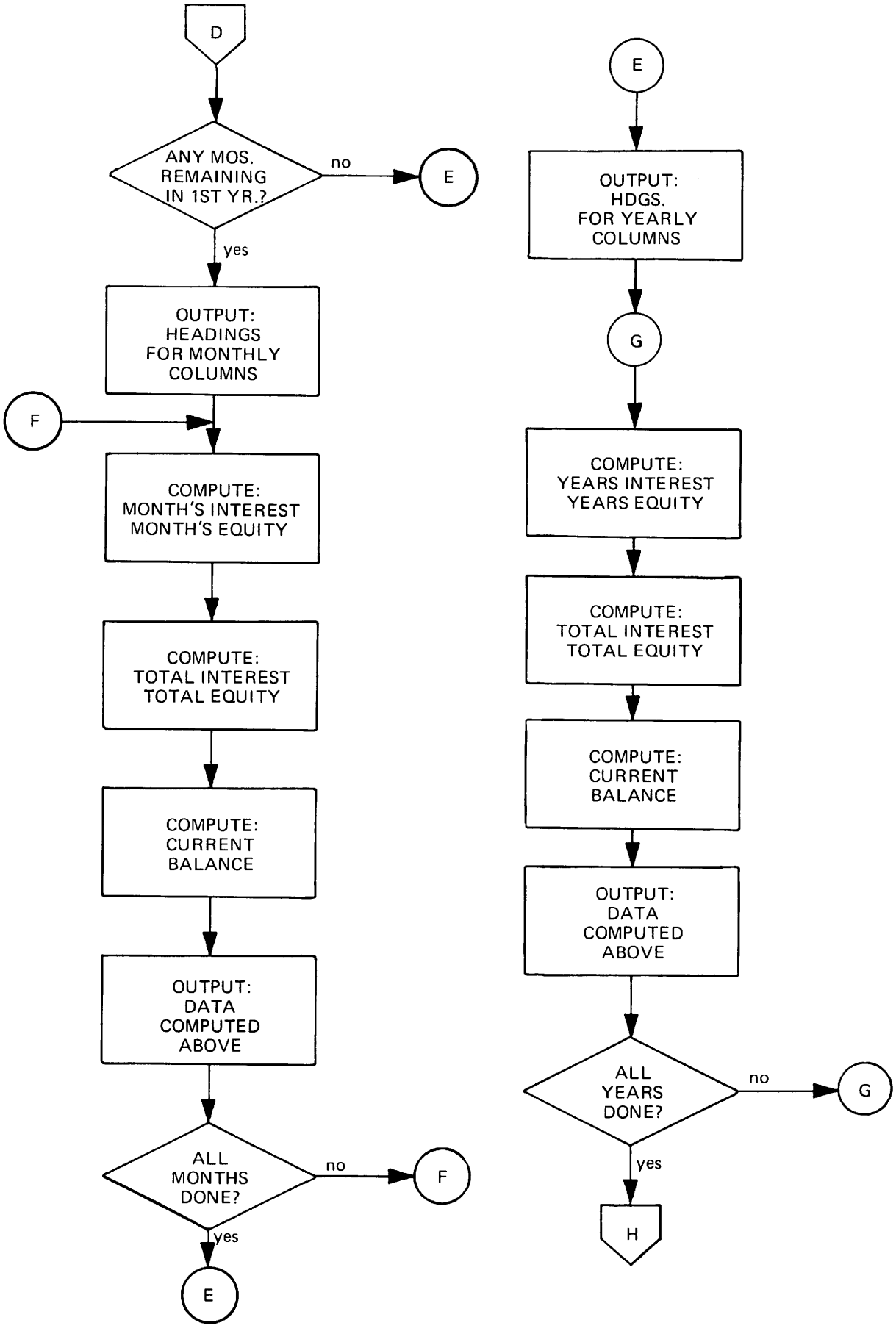
C. **Output:**

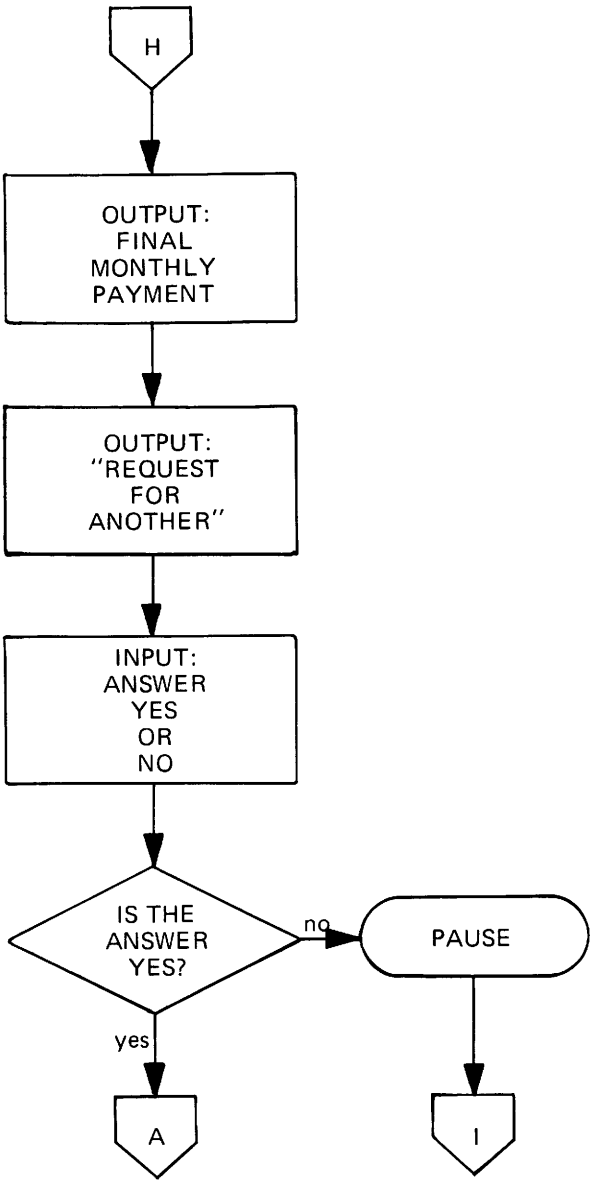
1. Requests for data.
   a. Instructions
   b. Number of months remaining in year
   c. Original debt
   d. Interest rate
   e. Mortgage type
   f. Year mortgage payments begin
2. Minimum down payment
3. Monthly payment
4. Interest, equity, title interest, title equity, principal, for each month
5. Interest, equity, title interest, title equity, principal, for each year
6. The month final payment is made
7. Request to compute more data sets

2. Flowchart


START
I
OUTPUT: "DO YOU NEED INSTRUCTIONS"?
INPUT: "YES" OR "NO"
IS THE ANSWER "YES"?
no
A
yes
OUTPUT: INSTRUCTIONS
A
OUTPUT: DATA REQUESTS
INPUT: NO. MOS. 1ST YR. DEBT INT. RATE MORT. TYPE 1ST YR.
J
IS THE NO. OF MONTHS IN THE FIRST YR. >12?
yes
C
no
B
C
OUTPUT: REQUEST FOR NO. OF MOS. 1ST YR.
INPUT: NO. MOS. 1ST YR.
J


B
COMPUTE: MONTHLY INTEREST RATE I = I/12
LOAN TYPE?
Savings & Loan
Commercial
F.H.A.
COMPUTE: DOWN PAYT. (.20*PRIN.)
COMPUTE: DOWN PAYT. (.25*PRIN.)
COMPUTE: DN. PAYT. = (.03*15K)+(.10*5K)+(.20 BAL)
COMPUTE: MONTHLY PAYMENT 2
COMPUTE: MONTHLY PAYMENT 3
COMPUTE: MONTHLY PAYMENT 1
OUTPUT: NEW PRINCIPAL & MON. PAYT.
COMPUTE: NEW PRIN. = PRIN-DN. PAYT.
COMPUTE: EQUITY
D


D
ANY MOS. REMAINING IN 1ST YR.?
no
E
yes
OUTPUT: HEADINGS FOR MONTHLY COLUMNS
F
COMPUTE: MONTH'S INTEREST MONTH'S EQUITY
COMPUTE: TOTAL INTEREST TOTAL EQUITY
COMPUTE: CURRENT BALANCE
OUTPUT: DATA COMPUTED ABOVE
ALL MONTHS DONE?
no
F
yes
E
E
OUTPUT: HDGS. FOR YEARLY COLUMNS
G
COMPUTE: YEARS INTEREST YEARS EQUITY
COMPUTE: TOTAL INTEREST TOTAL EQUITY
COMPUTE: CURRENT BALANCE
OUTPUT: DATA COMPUTED ABOVE
ALL YEARS DONE?
no
G
yes
H


H
OUTPUT:
FINAL
MONTHLY
PAYMENT
OUTPUT:
"REQUEST
FOR
ANOTHER"
INPUT:
ANSWER
YES
OR
NO
IS THE
ANSWER
YES?
no
PAUSE
yes
A
I

3. CAL Code And Sample Execution

```
-CAL
STATEMENTS = 300

HEADING =

>LOAD
FROM /MORTGAGE/

>TYPE ALL


1.0 TYPE "        THIS IS A PROGRAM TO COMPUTE STANDARD MORTGAGE DATA."
1.1 DEMAND IN FORM 1: A
1.2 TO PART 2 IF A#1
1.3 TYPE "THIS PROGRAM COMPUTES THE DOWN PAYMENT, MONTHLY PAYMENT,"
1.4 TYPE "INTEREST PAID, EQUITY ACCUMULATED, AND THE NEW BALANCE FOR"
1.5 TYPE "A STANDARD MORTGAGE.  COMPUTATION WILL FIRST BE PRINTED"
1.6 TYPE "FOR THE MONTHS REMAINING IN THE FIRST YEAR, THEN FOR EACH"
1.62 TYPE "YEAR, AND FINALLY THE PROGRAM WILL PRINT THE MONTH IN WHICH"
1.64 TYPE "THE FINAL MONTHLY PAYMENT WAS MADE."
1.66 LINE
1.68 TYPE "THE USER MUST INPUT CERTAIN PERTINENT DATA WHEN REQUESTED"
1.70 TYPE "BY THE PROGRAM.  WHEN THE MORTGAGE TYPE IS REQUESTED"
1.72 TYPE "IT IS ASSUMED BY THE PROGRAM THAT AN F.H.A. MORTGAGE"
1.74 TYPE "IS AMORTIZED OVER A 30 YEAR PERIOD, A CONVENTIONAL MORTGAGE"
1.76 TYPE "IS AMORTIZED OVER A 25 YEAR PERIOD WITH A MINIMUM DOWN PAYMET
1.78 TYPE "OF 20%, AND THAT A COMMERCIAL MORTGAGE IS AMORTIZED OVER"
1.80 TYPE "A 20 YEAR PERIOD WITH A MINIMUM DOWN PAYMENT OF 25%."
1.82 LINE

2.0 DEMAND IN FORM 2: R,P,I
2.1 DEMAND IN FORM 3: L,Y
2.2 TO PART 10 IF R>12
2.3 R1=12-R
2.4 I=I/1200
2.5 N=IF L=1 THEN 360 ELSE IF L=2 THEN 300 ELSE IF L=3 THEN 240
2.6 DO PART (L*20)
2.7 P=P-D
2.8 E1=D
2.9 M=(P*I*(I+1)↑N)/((I+1)↑N-1)

3.0 N=N-13
3.1 TYPE IN FORM 4: D,M
3.2 TYPE "
            TOTAL          TOTAL          MONTH'S          MONTH'S          CURRENT
MONTH      INTEREST       EQUITY         INTEREST         EQUITY           DEBT "
IF R#0
3.3 I1=I2=I3=E2=E3=0
3.4 DO PART 4 FOR K=R BY -1 TO 1
3.5 TYPE "
```

```
            TOTAL          TOTAL        YEAR'S         YEAR'S        CURRENT
YEAR       INTEREST       EQUITY      INTEREST       EQUITY         DEBT "
3.6 DO PART 5 FOR K=N BY -1 TO 0
3.7 DO PART 6 FOR K=R1 BY -1 TO 1
3.8 TO PART 7

4.0 I2=P*I
4.1 E2=M-I2
4.2 I1=I1+I2
4.3 E1=E1+E2
4.4 P=P-E2
4.5 TYPE IN FORM (200+K): I1,E1,I2,E2,P

5.0 I2=P*I
5.1 E2=M-I2
5.2 I1=I1+I2
5.3 E1=E1+E2
5.4 P=P-E2
5.5 I3=I3+I2
5.6 E3=E3+E2
5.7 DO PART 9 IF K MOD 12 = 0

6.0 I2=P*I
6.1 E2=M-I2
6.2 I1=I1+I2
6.3 E1=E1+E2
6.4 P=P-E2
6.5 I3=I3+I2
6.6 E3=E3+E2

7.0 Y=Y+1
7.1 TYPE IN FORM 5: Y,I1,E1,I3,E3,P
7.2 TYPE IN FORM 9:
7.3 TYPE IN FORM 100+R1:
7.4 LINE FOR K= 1 TO 6
7.5 DEMAND IN FORM 6: A
7.6 TO PART 2 IF A#0
7.7 LINE FOR K= 1 TO 6
7.8 PAUSE
7.9 TO PART 1

9.0 Y=Y+1
9.1 TYPE IN FORM 5: Y,I1,E1,I3,E3,P
9.2 I3~E3~0

10.0 TYPE"
          THE NUMBER OF MONTHS REMAINING IN ONE YEAR CANNOT
          EXCEED TWELVE."
10.1 DEMAND IN FORM 7: A
10.2 TO STEP 2.2

20.0 D=IF P>2E4 THEN ((P-2E4)*.20+950) ELSE IF P>15E3 THEN
       ((P-15E3)*.10+450) ELSE (.03*P)
```

```
40.0 D=.20*P

60.0 D=.25*P

FORM 1:
      DO YOU NEED INSTRUCTIONS ? (TYPE 1 FOR YES, OR 0 FOR NO) #
FORM 2:

         THE NUMBER OF MONTHS IN THE FIRST YEAR = #
         THE ORIGINAL DEBT = $#
         INTEREST RATE = #
FORM 3:
         TYPE OF MORTGAGE (1=FHA, 2=CONVENTIONAL, 3=COMMERCIAL) #
         MORTGAGE PAYMENTS BEGIN IN 19#
FORM 4:
         MINIMUM DOWN PAYMENT = $%%%%%.%%
         MONTHLY PAYMENT = $%%%%.%%
FORM 5:
19%%     $%%%%%.%%    $%%%%%.%%   $%%%%%.%%      $%%%%%.%%    $%%%%%.%%
FORM 6:
         DO YOU WISH TO COMPUTE DATA FOR ANOTHER MORTGAGE ?
         (TYPE 1 FOR YES, OR 0 FOR NO) #
FORM 7:
         THE NUMBER OF MONTHS IN THE FIRST YEAR = #
FORM 9:


         THE FINAL MONTHLY PAYMENT IS IN %
FORM 101:
JANUARY
FORM 102:
FEBRUARY
FORM 103:
MARCH
FORM 104:
APRIL
FORM 105:
MAY
FORM 106:
JUNE
FORM 107:
JULY
FORM 108:
AUGUST
FORM 109:
SEPTEMBER
FORM 110:
OCTOBER
FORM 111:
NOVEMBER
FORM 112:
DECEMBER
```

```
FORM 201:
 DEC.      $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%
FORM 202:
 NOV.      $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%
FORM 203:
 OCT.      $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%
FORM 204:
 SEP.      $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%
FORM 205:
 AUG.      $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%
FORM 206:
 JUL.      $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%
FORM 207:
 JUN.      $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%
FORM 208:
 MAY       $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%
FORM 209:
 APR.      $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%
FORM 210:
 MAR.      $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%
FORM 211:
 FEB.      $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%
FORM 212:
 JAN.      $%%%%%.%%     $%%%%%.%%     $%%%%%.%%      $%%%%%.%%     $%%%%%.%%

>TO PART 1
        THIS IS A PROGRAM TO COMPUTE STANDARD MORTGAGE DATA.
        DO YOU NEED INSTRUCTIONS ? (TYPE 1 FOR YES, OR 0 FOR NO) 1


THIS PROGRAM COMPUTES THE DOWN PAYMENT, MONTHLY PAYMENT,
INTEREST PAID, EQUITY ACCUMULATED, AND THE NEW BALANCE FOR
A STANDARD MORTGAGE.  COMPUTATION WILL FIRST BE PRINTED
FOR THE MONTHS REMAINING IN THE FIRST YEAR, THEN FOR EACH
YEAR, AND FINALLY THE PROGRAM WILL PRINT THE MONTH IN WHICH
THE FINAL MONTHLY PAYMENT WAS MADE.

THE USER MUST INPUT CERTAIN PERTINENT DATA WHEN REQUESTED
BY THE PROGRAM.  WHEN THE MORTGAGE TYPE IS REQUESTED
IT IS ASSUMED BY THE PROGRAM THAT AN F.H.A. MORTGAGE
IS AMORTIZED OVER A 30 YEAR PERIOD, A CONVENTIONAL MORTGAGE
IS AMORTIZED OVER A 25 YEAR PERIOD WITH A MINIMUM DOWN PAYMENT
OF 20%, AND THAT A COMMERCIAL MORTGAGE IS AMORTIZED OVER
A 20 YEAR PERIOD WITH A MINIMUM DOWN PAYMENT OF 25%.


        THE NUMBER OF MONTHS IN THE FIRST YEAR = 10

        THE ORIGINAL DEBT = $32500.00

        INTEREST RATE = 6.5


        TYPE OF MORTGAGE (1=FHA, 2=CONVENTIONAL, 3=COMMERCIAL) 2
```

MORTGAGE PAYMENTS BEGIN IN 1968

MINIMUM DOWN PAYMENT = $ 6500.00
MONTHLY PAYMENT = $ 175.55

MONTH	TOTAL INTEREST	TOTAL EQUITY	MONTH'S INTEREST	MONTH'S EQUITY	CURRENT DEBT
MAR.	$ 140.83	$ 6534.72	$ 140.83	$ 34.72	$25965.28
APR.	$ 281.48	$ 6569.63	$ 140.65	$ 34.91	$25930.37
MAY	$ 421.93	$ 6604.73	$ 140.46	$ 35.10	$25895.27
JUN.	$ 562.20	$ 6640.01	$ 140.27	$ 35.29	$25859.99
JUL.	$ 702.28	$ 6675.49	$ 140.07	$ 35.48	$25824.51
AUG.	$ 842.16	$ 6711.16	$ 139.88	$ 35.67	$25788.84
SEP.	$ 981.85	$ 6747.03	$ 139.69	$ 35.86	$25752.97
OCT.	$ 1121.34	$ 6783.09	$ 139.50	$ 36.06	$25716.91
NOV.	$ 1260.64	$ 6819.34	$ 139.30	$ 36.25	$25680.66
DEC.	$ 1399.75	$ 6855.79	$ 139.10	$ 36.45	$25644.21

YEAR	TOTAL INTEREST	TOTAL EQUITY	YEAR'S INTEREST	YEAR'S EQUITY	CURRENT DEBT
1969	$ 3053.28	$ 7308.91	$ 1653.53	$ 453.11	$25191.09
1970	$ 4676.47	$ 7792.37	$ 1623.19	$ 483.46	$24707.63
1971	$ 6267.27	$ 8308.20	$ 1590.81	$ 515.84	$24191.80
1972	$ 7823.54	$ 8858.59	$ 1556.26	$ 550.38	$23641.41
1973	$ 9342.94	$ 9445.83	$ 1519.40	$ 587.24	$23054.17
1974	$10823.01	$10072.41	$ 1480.07	$ 626.57	$22427.59
1975	$12261.12	$10740.94	$ 1438.11	$ 668.54	$21759.06
1976	$13654.46	$11454.25	$ 1393.34	$ 713.31	$21045.75
1977	$15000.02	$12215.33	$ 1345.57	$ 761.08	$20284.67
1978	$16294.62	$13027.39	$ 1294.59	$ 812.05	$19472.61
1979	$17534.83	$13893.82	$ 1240.21	$ 866.44	$18606.18
1980	$18717.01	$14818.29	$ 1182.18	$ 924.46	$17681.71
1981	$19837.28	$15804.66	$ 1120.27	$ 986.38	$16695.34
1982	$20891.49	$16857.10	$ 1054.21	$ 1052.44	$15642.90
1983	$21875.21	$17980.02	$ 983.73	$ 1122.92	$14519.98
1984	$22783.74	$19178.14	$ 908.52	$ 1198.12	$13321.86
1985	$23612.02	$20456.51	$ 828.28	$ 1278.36	$12043.49
1986	$24354.69	$21820.49	$ 742.67	$ 1363.98	$10679.51
1987	$25006.00	$23275.81	$ 651.32	$ 1455.33	$ 9224.19
1988	$25559.86	$24828.61	$ 553.85	$ 1552.79	$ 7671.39
1989	$26009.72	$26485.39	$ 449.86	$ 1656.79	$ 6014.61
1990	$26348.62	$28253.14	$ 338.90	$ 1767.74	$ 4246.86
1991	$26569.13	$30139.27	$ 220.51	$ 1886.13	$ 2360.73
1992	$26663.33	$32151.72	$ 94.19	$ 2012.45	$ 348.28
1993	$26666.16	$32500.00	$ 2.83	$ 348.28	$ 0.00

THE FINAL MONTHLY PAYMENT IS IN FEBRUARY

DO YOU WISH TO COMPUTE DATA FOR ANOTHER MORTGAGE ?
(TYPE 1 FOR YES, OR 0 FOR NO) 0

PAUSE IN STEP 7.8:
>

APPENDIX 1
PRECEDENCE LIST

This appendix gives the complete precedence list for the CAL operators and functions. Operators with the same precedence are performed from left to right. Parentheses may be used to alter precedence.

Precedence	Operator
1	– (unary minus, the negative sign)
2	↑
3	/ * MOD
4	The CAL functions
5	+ –
6	= # < <= > >= relational operators
7	NOT
8	AND OR
9	← (replacement)
10	= replacement operation

APPENDIX 2
CAL SUMMARY

NUMBERS

Integer (without decimal point)	For example; 357940
Decimal (with decimal point)	For example; 35.7940
Scientific Notation	For example; 3.57E23 (where E23 means 10^{23})

VARIABLES

Legal Variables	A - Z and A0 - Z9
Subscripted Variables	A(1), B(I-3*S), A6(N, M,...R)

ARITHMETIC OPERATORS

In order of priority

–	Unary Minus (Negation)
↑	Exponentiation
*, /, MOD	Multiplication, Division, Modulo
+, –	Addition, Subtraction
←	Replacement

RELATIONAL OPERATORS

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

LOGICAL OPERATORS

AND	Logical Multiplication
OR	Logical Addition
NOT	Reverses Logical Value

MATHEMATICAL FUNCTIONS (STANDARD)

ABS (A)	Absolute value of A
SIN (A)	Sine of A
COS (A)	Cosine of A
TAN (A)	Tangent of A
ATAN (X,Y)	Arctangent in radians, over the range $-\pi$ to $+\pi$ of X/Y.
EXP (A)	e to the power A
LOG (A)	Natural logarithm of A
LOG10 (A)	Base 10 logarithm of A
SQRT (A)	Positive square root of A
IP (A)	Integer part of A
FP (A)	Fractional part of A

ITERATIVE FUNCTIONS

SUM
PROD
MAX
MIN

(Any form of FOR clause without the word FOR:expression)

Example

SUM(variable = limit BY interval [TO limit: / UNTIL condition:expression) / WHILE condition:]

PROGRAMMER DEFINED FUNCTIONS

DEFINE function name [parameter list] = expression

DEFINE function name [parameter list] :TO STEP step number

step number RETURN expression

FORM STATEMENTS

FORM form number: ↵

INPUT SPECIFICATION

For any number type #

OUTPUT SPECIFICATION

Integer	%%%%%%
Decimal	%%%%%%.%%
Exponential	###### *(Minimum 6)*

Any other characters (blanks included) printed as shown.

COMMANDS

INPUT/OUTPUT COMMANDS (DIRECT OR INDIRECT)

DEMAND variable list
DEMAND IN FORM form number:variable list
TYPE variable list
TYPE IN FORM form number:variable list
TYPE "text"
TYPE STEP step number
TYPE PART part number
TYPE FORM form number
TYPE function name
TYPE ALL STEPS
TYPE ALL FORMS
TYPE ALL FUNCTIONS
TYPE ALL VALUES
TYPE ALL

DATA FILES (DIRECT OR INDIRECT)

OPEN /file name/ FOR INPUT *or* OUTPUT AS FILE file number

READ INPUT FROM file number:variable list

WRITE ON file number:variable list

WRITE ON file number IN FORM form number:variable list

OUTPUT ON file number:non-subscripted variable list

CLOSE file number

REPLACEMENT COMMAND (DIRECT OR INDIRECT)

single variable = expression For example, A1 = PI*R↑2

CONTROL COMMANDS

Direct Or Indirect

TO PART step or part number
TO STEP step number
DO PART step or part number
DO STEP step number

Indirect Only

PAUSE
DONE

Direct Only

RUN
GO
QUIT
STEP

Miscellaneous

Direct Or Indirect

LINE
PAGE
$
! Comments

Direct Only

LINES

Program Files (Direct Only)

DUMP ↵
TO /file name/
LOAD ↵
FROM /file name/

DELETE COMMANDS (DIRECT ONLY)

DELETE STEP step number
DELETE step number
DELETE PART step or part number
DELETE FORM form number
DELETE variable
DELETE function name
DELETE ALL STEPS
DELETE ALL FORMS
DELETE ALL VALUES
DELETE ALL FUNCTIONS
DELETE ALL *or* CLEAR

EDIT COMMANDS (DIRECT ONLY)

EDIT STEP step number
EDIT step number
EDIT FORM form number
EDIT function name
MOD STEP step number
MOD step number
MOD FORM form number
MOD function name

MODIFIERS

IF expression
UNLESS expression
UNTIL terminating condition
WHILE terminating condition
FOR variable = list of values
FOR variable = limit BY interval [TO limit | UNTIL condition | WHILE condition]
WHERE expression & expression
IF condition THEN expression ELSE expression

INDEX

NOTE: Page numbers which appear in **bold** *face type refer to those pages where the listed item receives the most detailed discussion.*

TYMSHARE MANUALS

Instant Series

CAL
SUPER BASIC
EDITOR

Reference Manuals

EXECUTIVE
CAL
SUPER BASIC
EASYPLOT
EDITOR
FORTRAN IV
FORTRAN II
LIBRARY
COGO
ECAP
ARPAS/DDT
BRS

TYMSHARE®

TYMSHARE, INC., 525 University Avenue, Suite 220, Palo Alto, California 94301

SAN DIEGO/ORANGE COUNTY
4630 Campus Drive, Suite 209
Newport Beach, California 92660
Telephone: 714/540-5940

NEW YORK
464 Hudson Terrace
Englewood Cliffs, New Jersey 07632
Telephone: 201/567-9110

DALLAS
2355 Stemmons Bldg., Suite 1010
Dallas, Texas 75207
Telephone: 214/638-5680

SAN FRANCISCO
745 Distel Drive
Los Altos, California 94022
Telephone: 415/961-0545

LOS ANGELES
336 East Kelso Street
Inglewood, California 90301
Telephone: 213/677-9142

SEATTLE
2200 6th Avenue, Suite 810
Seattle, Washington 98121
Telephone: 206/MA 3-8321

WASHINGTON, D.C.
1911 N. Fort Myer Drive, Suite 907
Arlington, Virginia 22209
Telephone: 703/524-5930