Students' CAL Manual

by

John P. Gilbert

Harvard Computing Center

June, 1967

CONTENTS

CHAPTER 1: LOGGING ON AND OFF

In order to distinguish between what the user types and what the computer types I shall underline the output from the computer. I shall also need to refer to three non-printing keys on the teletype. They are the (escape), (line feed), and the (return) keys. The (line feed) and (return) keys are located on the right-hand side of the keyboard. The (escape) key is on the left side.

Pressing the (return) key signals the computer that you are through with that line of typing and want to start a new one; the print drum will return to the left margin and the paper will space. (line feed) continues the statement you are typing onto the next line. (escape) is used to say "forget what has been typed and start the line again."

Now let us get on the machine. The first step is to connect the teletype to a computer port. The steps for this are: 1) Press the originate button, labelled ORG, which is below the dial and to the left--it should light up. 2) After you hear the dial tone on the speaker under the buttons, use the dial on the teletype to dial the computer. Dial 2 on teletypes which are extensions of the Harvard switch and 491-5900 on those connected to outside lines. A small knob controls the volume of the speaker but normally it should not need adjustment. 3) You should hear the teletype ringing the port and then a beep when the port answers. 4) The computer will then type:

```
HARVARD TIME SHARING SYSTEM (D00-H05): 6-12-67 (return)
JUNE 28, 1967 3:58 p.m.  (return)

ACCOUNT:
```

We must now convince the computer that we are honest bill paying customers. To do this we must have an account number, a password and a user name. For the purposes of illustration let us suppose that our account number is 123, our password is ABCDEF, and our name is LEGION. The letters of the password will not be printed. What is shown below is what must be typed. A correct log in would go as follows:

```
HAVARD TIME SHARING SYSTEM ( 00-H05): 6-12-67 (return)
JUNE 28, 1967 3:58 P.M.  (return)
ACCOUNT 123(return)
PASSWORD: ABCDEF(return)
NAME: LEGION.(return)
@
```

Note that the computer provided all the (returns). If you make a mistake in this process the computer will type out

"INVALID USER" and then start the process of identification all over again. When you have successfully entered the computer will type a @ sign. This character is the trademark of the executive program. The EXEC has a number of subsystems, that it can provide. We wish the one called CAL so we ask for it. CAL will want some information and the dialogue should go like this:

```
@CAL.(return)
CAL 1.69 Number OF STATEMENTS NEEDED = 20(return)
HEADING, PLEASE (return)
DEMONSTRATION PROGRAM (return)




DEMONSTRATION PROGRAM PAGE 1 (return)
≥
```

Note that when CAL asks for the number of statements you must give a number and then provide the (return) to tell CAL that you are through typing. The same is true of the title. CAL starts you off with a new page complete with title and page number and then gives you a line starting with >. > is the trademark of CAL. You can always tell who you are talking to by the symbol starting each line. Do not ask for too many statements or you will not have any room for data!

Let us assume that you are tired of CAL, or have even finished a problem and want out. The procedure for logging out properly is as follows: first get back to the EXEC; this is done by pushing (escape) several times. You can tell when you have succeeded because the computer will type @. Second you type LOG for LOGOUT LEGION and then a period. It should look like this:

```
≥(return)
≥(return)
@LOGOUT LEGION.  (return)
JUN 28, 1967.  3:59 P.M.  (return)
TIME USED (return)
 PU: 00:00:01 (return)
CONNECT: 00:02 (return)
The computer turns off the teletype.
```

SUMMARY: ENTERING AND LOGGING OUT

Let us review this process by entering, doing nothing, and logging out again.

You press (originate) to get dial tone;

dial 2 or 491-5900 as appropriate.

```
HARVARD TIME SHARING SYSTEM (D00-H05): 6-12-67 (return)
JUN 28, 3:58 P.M.  (return)
ACCOUNT: 123 (return)
```

```
PASSWORD: ABCDEF (return)
NAME: LEGION. (return)
@CAL. (return)
CAL 1.69 NUMBER OF STATEMENTS NEEDED = 20 (return)
HEADING PLEASE (return)
DEMONSTRATION PROGRAM (return)
```

```
DEMONSTRATION PROGRAM (return)
≥(escape)
≥(escape)
@LOGOUT LEGION. (return)
JUN 28, 1967 3:59 P.M.  (return)
TIME USED (return)
CPU: 00:00:00 (return)
CONNECT: 00:01 (return)
The computer turns off the teletype.
```

You can tell that you have logged out successfully by the time used message which refers to the time used by the central processor and the amount of time you have been connected. If you have logged out correctly the computer will turn off your teletype for you. If LOGOUT does not work try EXIT. If that does not work call the Computing Center! ext. 3272 on a telephone.

CHAPTER 2

I shall now assume that you have entered and got to CAL as outlined in Chapter 1. In particular I hope that you are looking at a greater-than sign.

CAL has two modes of operating. One of these, called the direct mode, acts like a desk calculator. The other, called indirect, utilizes more of the computer's facilities. Almost all CAL commands can be used in either mode except for a few which only make sense in one. We shall start out in the direct, or desk calculator, mode, but all the commands I will mention can be used in either mode. A distinctive characteristic of the direct mode is that as soon as you have finished typing a command and type (return), the computer will execute the command. When it has finished it will start a fresh line with a > sign.

2.1 The Command TYPE

The first command you will need is the command TYPE followed by whatever you would like typed and then a (return). Here are some examples:

```
>TYPE 34/2 (return)
______34/2 =____17 (return)
>TYPE 4*6, 3/4 (return)
______4*6 =_____24 (return)
______3/4 =_____0.75000000 (return)
>
```

2.2 The operations +, -, *, /, ↑, and their precedence

The arithmetic operators are +, -, *, /, and ↑. They stand for addition, subtraction, multiplication, division, and exponentiation (raising to a power) respectively. In evaluating an arithmetic expression CAL will do the exponentiations first, then the multiplications and divisions (working from left to right), and finally the additions and subtractions. If this is not the order desired, or if there is some doubt, parentheses ( ) should be used to make the expression unambiguous. The following examples illustrate some of these rules.

These examples illustrate the precedence of operators:

```
≥           TYPE 4+3↑2                    exponentiation first
 4+3↑2      =        13
≥           TYPE (4+3)↑2
 (4+3)↑2    =        49
```

These illustrate that ↑ is done before + unless ( ) are used

```
≥           TYPE 4*3↑2
 4*3↑2      =        36
≥           TYPE 4*(3↑2)
 4*(3↑2)    =        36
≥           TYPE (4*3)↑2
 (4*3)↑2    =       144
```

These illustrate that ↑ is done before *

```
≥           TYPE 2*3/4*5                   left to right rule
 2*3/4*5        =       7.5000000
≥           TYPE (2*3)/(4*5)
 (2*3)/(4*5)  =        0.30000000
≥           TYPE ((2*3)/4)*5
((2*3)/4)*5   =         7.5000000
```

2.3 The command SET

The SET command is used to store a number, or the result of a calculation, for future use. I will explain just what this means after giving some examples:

```
≥ SET A = 3 (return)
≥ SET B = 4 (return)
≥ SET C = A*B (return)
≥ TYPE C-2 (return)
__C-2 = 10 (return)
≥
```

The value of the number or the expression on the right hand side of the equal sign is stored for later reference by the machine as the value of the variable on the left hand side. Variables can only have single letters or a single letter followed by a single digit in their names but they can be subscripted with great abandon. The subscripts are placed in parentheses and separated with commas. A, A(0), A1, A(1), and A(B(C,D),E,4) are all different and acceptable variables. The last will have meaning only if the variables C, D, B(C,D), and E have already been SET. It is important to realize that the SET command does not represent an equation. For example:

```
≥ SET A = A + 1
```

is a perfectly good and legal statement meaning replace the old value of A by a new value one greater. What a SET command does is to instruct the computer to first evaluate the expression on the right of the equal sign and then to store the result as the value of the variable on the left. The old value of the variable is lost. To the computer a variable is the name of a place where a number can be stored. The command

```
≥ SET A = B + C (return)
```

is a way of saying "to the number stored in B add the number stored in C and store the result in A", where A, B, and C are names for actual storage locations in the computers memory.

<u>WARNING:</u>

The command SET 2*A = B + C has no meaning to CAL because it does not have a variable 2*A in which to store the result of adding the contents of B to that of C.

<u>Spaces</u>

All the command words, or reserved words, such as SET, need to be separated from their surroundings by a space. The only exception to this is that they don't need a space at the beginning of a line when they are next to ≥. Aside from this CAL is quite indifferent to spaces and they can be used as desired.

The word SET can be leftout of a SET command. Thus A = B is identical to SET A = B.

2.4 Corrections

By now you may be wondering how to correct a mistake in typing a command. If you push (escape) the computer will ignore what you have typed and start you out on a fresh line, where you can try again. If you push (return) the computer will try to execute what you have typed and if it doesn't succeed it will type a snide remark and start you out with a fresh line. In addition to these two courses of action CAL provides a whole series of editing commands but, for the moment, I will discuss only two of these. All editing commands are given by holding the (control) key down while typing another key. The (control) key is located on the left side of the keyboard. The two commands are (control)A and (control)W.

2.5 Control A

(control)A is given by holding the (control) key down while typing A. The effect of this is to print ↑ and to delete the last character typed. This deleted character could of course have been a blank. To delete several characters hold (control) down and type as many A's as there are characters to be deleted; the computer will print a ↑ for each character deleted: These ↑'s are only printed and they do not become part of the statement being formed in the computer.

2.6 Control W

(control)W acts like (control)A except that it deletes the preceding word, and the blank in front of it. To indicate that it has done this it prints\.

<u>Escape</u>

One word of caution, if you push (escape) twice in succession you will find yourself talking to the EXEC (with a @ instead of a >). To get back to CAL without losing what you have done, type CON<u>TINUE CAL</u>.; note that you type the period to confirm that you wish to do what was typed. You can tell if this has succeeded by whether or not you get a line starting with a >.

CHAPTER 3

In the indirect mode a number of commands are organized into a program, which is then run as a unit, rather than having each command executed as it is typed. In order to write a program one must not only specify the individual commands but also the order in which they are to be executed. To make this possible each command, or statement, is given a STEP number. Indeed, CAL determines for every statement whether it is to be executed immediately (direct mode) or is to be incorporated into a program (indirect mode) by whether or not it begins with a STEP number.

3.1 Numbering Steps

To help in organizing the program, statements are numbered on two levels by a pair of numbers separated by a decimal point. The integer (to the left of the point) is the PART number. The fraction (to the right of the decimal) serves to order statements within each PART. As numbered statements are typed into the computer CAL orders them by their STEP numbers, considered as decimal numbers. For example, STEP 1.2 is before STEP 1.245 which is before STEP 2.001, no matter in what order the programmer typed them. The first two would be in PART 1 while the third is in PART 2. This system of numbering allows you to insert a command between two which have already been typed. For example, if 1.3 and 1.4 are already used, you might insert 1.35.

In the absence of any directions to the contrary the computer will execute the statements of a program in their numerical order. The programmer can, however, direct the computer to deviate from this order in a variety of ways.

Let us look at an example:

```
>1.1 SET J = 1 (return)
>1.2 TYPE J↑2 (return)
>1.3 SET J = J + 1 (return)
>1.4 TO STEP 1.2 (return)
```

This example contains two new things. 1) the steps are numbered and we see that all four steps belong to PART 1. In addition the computer will treat them as being ordered in the order shown. 2) STEP 1.4 contains the new command TO. This has the effect of sending the program to the step specified, here 1.2, which it will execute and then proceed to the steps following this new starting point. The program above will go around and around forever, caught in an infinite loop, because every time it gets to STEP 1.4 it will be sent back to STEP 1.2. Suppose we had started this program running by typing the direct command "TO STEP 1.1 (return)". What could we do to stop the unending sequence of squares which this program would produce? The answer is that any program can be interrupted by pushing the (escape) key. It may take a second for the machine to stop in some programs but never disconnect your teletype in this situation! If you hit (escape) too many times the worst that can happen is that you will end up talking to the EXEC and you can type CONTINUE CAL. to get back to your program. Don't forget the period.

3.2 IF modifier

Clearly it would be convenient if there were some way to tell the program when it should stop typing out more squares and proceed to the next part of the program. One way to do this is the IF modifier. I shall add it to the above example.

```
≥1.1 SET J = 1 (return)
≥1.2 TYPE J↑2
≥1.3 SET J = J + 1 (return)
≥1.4 TO STEP 1.2 IF J < 5 (return)
≥1.5 TYPE "DONE" (return)
≥TO STEP 1.1 (return)
_____J↑2 =  1
_____J↑2 =  4
_____J↑2 =  9
_____J↑2 = 16
DONE
≥
```

The effect of the IF is to execute the command in front of the IF if and only if the condition after the IF is true. If the condition is not true the computer ignores the part before the IF and goes on to the next statement of the program. In this example the computer jumps back to STEP 1.2 as long as J is less then 5. When J becomes equal to 5 the condition is no longer true and the computer ignores the command "TO STEP 1.2" and goes on to STEP 1.5, which is to type the literal DONE. Note the use of "s". Since there are no more commands to be executed after this one the computer stops, printing out ≥ as a sign that it is ready for more input.

This example illustrates a program loop or iteration. Since much of the power of the computer comes from its ability to perform many such iterations in a short time it is important to understand clearly how these loops work. Let us discuss the above example in detail. First notice the role played by the variable J. Let us call it the variable of iteration. Second, STEP 1.1 is only executed once when it initializes the variable of iteration i.e. starts it off, with the value 1. Third, STEP 1.3 increments J by a fixed amount, 1 in this case, every time around. Fourth, STEP 1.4 tests whether J has gotten so big that the computer should go on to the next part of the program. I have not said anything about STEP 1.2 because it does not have anything to do with determining the iteration. Any statement or statements could have been put in its place and they would have been done for J = 1, 2, 3, and 4. The four quantities which determine this type of iteration are the variable of iteration, its initial value, the amount it is incremented by, and the final value of J for which the iteration should be done.

The form of iteration illustrated in the preceding example occurs so often in writing programs that CAL has a special command for setting up this type of loop. I will do the same problem using this special command, which like the IF acts by modifying another command.

```
≥2.1 TYPE J↑2 FOR J = 1 BY 1 TO 4 (return)
≥2.2 TYPE "DONE" (return)
≥TO STEP 2.1 (return)
_____J↑2_=___1
_____J↑2_=___4
_____J↑2_=___9
_____J↑2_=__16
DONE
≥
```

When the computer comes to the word "FOR" it sets up a loop to operate on the preceding part of the statement. To be able to do this it must have the four things which specify an iteration. These are given to it by what follows the FOR in the form "variable of iteration = initial value BY increment TO final value". The last three quantities do not have to be numbers; they can be variables, which have been previously set, or even arithmetic expressions which the computer will evaluate when it comes to them. Because 1 is so often used as an increment the BY 1 may be omitted from a FOR clause and 1 will be assumed as the increment by CAL.

Some examples using FOR:

```
>TYPE J FOR J = 3 TO 7
     J =      3
     J =      4
     J =      5
     J =      6
     J =      7


>TYPE J,K FOR J = 1 BY 2 TO 4 FOR K = 10 TO 12
     J =      1
     K =     10
     J =      3
     K =     10
     J =      1
     K =     11
     J =      3
     K =     11
     J =      1
     K =     12
     J =      3
     K =     12
```

There are several points to notice about this last example.

1) For each value of K it went through the entire loop on J.

2) J did not take the value 4 because it was told to increase by twos from one so that it went 1, 3, 5 but 5 was too large so it stopped.

The following shows how this could be used to evaluate a function of two variables, namely J↑K.

```
≥TYPE J,K,J↑K FOR J=1 BY 2 TO 4 FOR K=10 TO 12
     J =          1
     K =         10
   J↑K =          1
     J =          3
     K =         10
   J↑K =      59049
     J =          1
     K =         11
   J↑K =          1
     J =          3
     K =         11
   J↑K =     177147
     J =          1
     K =         12
   J↑K =          1
     J =          3
     K =         12
   J↑K =     531441
≥
```

CHAPTER 4

At this point we have the basic tools necessary to solve a wide variety of programming problems. We can enter data by using the SET command in the direct mode. We can compute the value of complicated algebraic expressions, and we are able to do these calculations iteratively using the FOR construction. We can transfer control from one part of a program to another, using TO. The IF (condition) construction allows us to do these things sometimes and to skip them at others. Finally we can have the results of these computations typed out.

We do not, as yet, have much experience in how to use these tools to accomplish our aims, and until we do, we cannot appreciate how much can be accomplished with these commands. For the most part the rest of the commands in CAL simply provide easier or more elegant ways to achieve results which could have been obtained with the commands we have. In this chapter I shall introduce some of these labor saving devices and illustrate some common calculations.

4.1 Adding numbers

Let us consider the problem of adding up a series of numbers. If there were only a few we could write a program which would SET A equal to the first number, B equal to the second, etc. We could then SET X = A+B+... and then TYPE X. It is clear that we would not want to do this for very many numbers. We need a better way to get data into the computer and we need a better way to handle it once it is in. A solution to the first problem is provided by the DEMAND command (we shall find an even better way a little later). The command DEMAND X causes the computer to type out X = and then wait for you to type a number followed by a (return). The computer will then SET X equal to the number and proceed to the next step of the program. One can put a list of variables separated by commas, after DEMAND and the computer will ask for them one at a time. For example:

```
>1.1 SET S = 0  *

>1.2 SET X = 0

>1.3 SET S = S + X

>1.4 DEMAND X

>1.5 TO STEP 1.3 IF X < 9999

>1.6 TYPE S

TO STEP 1.1

      X = 56.89 (return)

      X = 33.65 (return)

      X = 18.93 (return)

      X = 48.25 (return)

      X = 999999 (return)          large number to terminate
                                      reading
      S =       157.72000
>
```

In this example S is the cumulative sum. Each new value of X replaces the old one which is then lost. This is an advantage if we are short of space in the computer and if we have no further need for the individual values. We use an abnormally large value of X to signal that we have entered all the numbers we wish to sum. Suppose we had wished to save

*From now on I will not indicate the non-printing characters such as (return) and (escape) except in special cases and I shall also only underline the computer output when it is necessary to clarify a particular point.

these numbers for some further calculation. The best way to do this is to put them into an array.

Here is one way this could be done.

```
>1.1 DEMAND N
>1.2 DEMAND D(I) FOR I = 1 TO N
>1.3 SET S = 0
>1.4 SET S = S + D(J) FOR J = 1 TO N
>1.5 TYPE S

>TO STEP 1.1

       N = 4
D(1)     = 56.89
D(2)     = 33.65
D(3)     = 18.93
D(4)     = 48.25
       S =       157.72000
>
```

This program first asks how many numbers it is to read and then proceeds to put these numbers into D(1), D(2),...D(N). Step 1.2 clears out any old value which might have been in S and 1.3 does the actual addition. If the program were interrupted in the middle of this loop S would contain the sum of the D(J)'s which had been indexed before the interruption.

Note that it makes no difference which letter we use as the index variable. When J = 7, D(J) is exactly the same as D(I) when I = 7. This point often causes confusion and you should remember that it is the numeric value of the index which determines which D in the array is intended, not the letter used.

4.2 SUM operation

Adding numbers is such a common occupation that CAL has a special command for it. The statement

```
SET S = SUM (J = 1 TO N : D(J) )
```

has exactly the same effect as steps 1.3 and 1.4 in the preceding example. There are four functions of this type and I will discuss them later in this chapter.

We will often find that we would like to include more than one program step in a FOR loop. We might wish to sum not only the quantities D(I) in the example above but also their squares. One way to accomplish this is to put all the steps to be included in the loop into a special PART and then use the DO command. I will illustrate assuming that the array D(I) has already been entered and that N has been SET. In particular I use the values from the previous example.

```
>1.1 SET S = S + D(I)
>1.2 SET Q = Q + D(I)↑2
>

>2.1 SET S ← Q ← 0
>2.2 DO PART 1 FOR I = 1 TO N
>2.3 TYPE S/N, (N*Q-S↑2)/N

>TO STEP 2.1
    S/N     =        39.4300000
(N*Q-S↑2)/N =      836.30240
```

In step 2.1 I have introduced the replacement arrow ←. This causes the value of Q to be SET to zero and then the value of S to be SET to zero. It can be used instead of the equal sign in any SET command.

4.3 DO commands

The effect of the DO is like that of the TO with one important difference. When the computer has finished the step or PART specified it does not continue on from that point but rather it goes back to the step which contained the DO and proceeds from that point. Thus STEP 2.2 causes the two steps in PART 1 to be done for each value of I and when the loop has been satisfied the computer goes on to STEP 2.3. If I had used TO instead of the DO the computer would have done STEPs 2.1, 2.2, PART 1, 2.1 again, 2.2 again, PART 1 etc. In addition each time it came to STEP 2.2 it would have tried to do it all over again from the beginning with I = 1. Note that we started the program at STEP 2.1 not 1.1. The DO makes it possible to write programs as a series of PARTs and then to have a master PART which controls the order and the number of times these PARTs are to be done. The DO does not have to be used in conjunction with a FOR clause; instead it can be used alone or with any of the other modifiers. You can DO a STEP as well as a PART and when you DO a PART the number of the PART can be specified by a variable or an expression.

4.4 FORMs and TYPE IN FORM

In order to label our output and to put more than one number on a line we use FORMs. A FORM allows us to specify just what will occupy each space on a line when it is typed out. In order to distinguish between different FORMs in the same program each is given a unique number. The FORM is used in conjunction with the TYPE IN FORM command, which specifies both which FORM is to be used, by giving its number, and what quantities are to be typed. The FORM specifies how these quantities are to appear. I shall give some examples and then discuss them.

```
>FORM 1 : (line feed)

X IN FIXED FORM = %%%%.%%   X IN FLOATING FORM = ########
>TYPE IN FORM 1 : 24.6, 24.6

X IN FIXED FORM = 24.60     X IN FLOATING FORM = 2.46 01
>
>FORM 2 : (line feed)

   %%%%%   %%%%%   %%%%%   %%%%%
>TYPE IN FORM 2 : I FOR I = 1 TO 10
>      1       2       3       4
       5       6       7       8
       9      10      >
```

Notice that the : is followed by a line feed and not a return. The computer provides a new line so that the FORM can be specified exactly as it is to be typed. When using % to define a field be sure to use an extra one for the sign of the number even if you know that the number will be positive. When using # to specify a field for numbers in floating point notation one must use five more #s than significant digits desired to provide room for sign, decimal point, and exponent (which may be negative). Numbers will be rounded on the right to fit the space provided. If there is not enough room to the left of the decimal in the field defined by %s an error message will be printed when the program comes to that point. If not all the numbers which a FORM was expecting are provided by the TYPE IN FORM command it will type what it can and then wait for the missing number. If a new TYPE IN FORM is then executed this new form will start where the other left off rather than on a new line. The number of the FORM in the TYPE IN FORM command can be specified by a variable or an expression as well as by a plain integer.

4.5 DEMAND IN FORM

A special use of DEMAND uses a FORM to greatly simplify the problem of entering large amounts of data. Let me illustrate the way this works.

```
>1.1 DEMAND IN FORM 1 : X(I) FOR I = 1 TO 6

>1.2 TYPE X(I) FOR I = 1 TO 6

>FORM 1 :

 # # #

>

>TO STEP 1.1

 123    234    34.56    45.67    42.35E3    42.35E-3

     X(1) =       123

     X(2) =      234

     X(3) =       34.560000

     X(4) =       45.670000

     X(5) =     42350

     X(6) =        4.2350000-02

>
```

Each number is terminated by a space except the last which gets a (return), to start a new line.

4.6 Floating point numbers

As this example shows, we can input a number in any combination of three formats--integer, decimal, or floating point. For those not familiar with floating point the exponent which follows the E indicates how many places the decimal point should be moved to obtain the correct value.

The decimal point is moved to the left if the exponent is negative and to the right if the decimal is positive. Thus 4.0E-10 is a very small number while 4.0E10 is a large number, 40,000,000,000 to be exact.

There are a number of mathematical functions which are built into CAL. These functions can appear as part of an expression and their arguments can also be expressions. In addition a function can be used as an argument of a function. When the argument of a function is an expression it must be enclosed in parenthesis. It is probably just as well to get into the habit of always using parenthesis. The built-in functions are:

4.7 BUILT-IN FUNCTIONS

ABS gives the value of the argument taken as positive.

ABS(-98) = 98

SIN is the trigonometric sine function, where the argument is the angle measured in radians.*

COS is the cosine with the argument in radians.*

TAN is the tangent argument again in radians.*

EXP is the exponential function i.e. e raised to the power of the argument.

---

*Radian measure is a way of measuring angles in terms of the length of the arc compared with the radius. 2(pi) radians = 360°. To convert from degrees to radians multiply by pi and divide by 180°.

LOG is the natural logarithm of the argument.

LOG10 is the logarithm to the base ten.

SQRT is the squareroot of the argument.

IP is the integer part of the argument, i.e.

IP(34.56) = 34

FP is the fractional part thus FP(34.56) = 0.56.

To facilitate working with radians CAL has the special constant

PI whose value is pi to eight significant figures.

4.8 Special Functions

In addition to the built-in functions, which have already been mentioned there are four rather unique functions in CAL; these are SUM, PROD, MAX, and MIN. The arguments of these functions contain a FOR clause, (the word FOR does not appear, however), and an expression. To illustrate the form of these functions suppose that we already have in the computer an array of numbers D(I) where I runs from 1 to N: and we executed the following program steps:

```
3.1 SET S = SUM(I = 1 TO N : D(I))
3.2 SET T = PROD(I = 1 TO N : D(I))
3.3 SET U = MAX(I = 1 TO N : D(I))
3.4 SET V = MIN(I = 1 TO N : D(I))
```

The result would be that S would equal the sum of the Ds, T would equal the product of all the Ds, U would equal the largest of the Ds and V would equal the smallest of them. The clause before the : has the same rules as that following a FOR, in particular it can be BY some increment other than one if desired. Any expression can follow the : .

4.9 Mean, variance or standard deviation

To show off our new powers let us write a program to find the mean, variance, and the standard deviation of a set of numbers. Although we don't need to for this program let us save the data in an array called D, for data. Rather than figuring out just how we are going to do everything before we write the program let us see if we can organize things as we go along.

First let us write a part to read the Ith datum and add it to the cumulative sum and the cumulative sum of squares. This whole part will be done over and over, once for each value of I.

```
>1.1 DEMAND IN FORM 1 : D(I)
>1.2 SET S = S + D(I)
>1.3 SET T = T + D(I)↑2
```

Now let us write a part which will, assuming that we have the sum of N observation in S and of their squares in T, compute their mean, variance, and standard deviation as required. We might as well print these out while we are at it.

```
>2.1 SET M = S/N
>2.2 SET V = (T - M*S)/N
>2.3 SET W = SQRT(V)
>2.4 TYPE IN FORM 2 : M,V,W,N
```

Finally we write a part which sets up the initial values and does the other parts in the correct sequence.

```
3.1 DEMAND N
3.2 SET S←T←0
3.3 DO PART 1 FOR I = 1 TO N
3.4 DO PART 2
```

We must not forget to specify the two forms!

```
FORM 1 :
#  #  #  #  #  #

FORM 2 :
MEAN = %%%%.%%%  VAR = %%%%%%.%%  SIGMA = %%%.%%%%  N = %%%%
```

We are now, I hope, ready to run the program.

```
>DO PART 3
   N = 10                   (Machine typed N=; you typed 10)
 1 2 3 4 5 6 7 8 9 10    (You typed these numbers)
MEAN = 5.500     VAR = 8.25     SIGMA = 2.8723     N = 10
```

Suppose now that we are reading an article in which the sum, the sum of squares, and the number of observations are given and we wish to check the later calculations. We don't want to change what we have so we add an additional part.

```
>4.1 DEMAND N,S,T
>4.2 DO PART 2

>DO PART 4

          N = 20
          S = 1234
          T = 2345678
MEAN = 61.70  VAR = 113477.01  SIGMA = 336.8635  N = 20
>
```

This is a modest example of the great flexibility which can be gained by dividing a problem into separate tasks each of which is done by a different part under the direction of a control program. Indeed, breaking a large problem down into subproblems is one of the most important techniques of programming.

CHAPTER 5: MODIFIERS, LOGICAL EXPRESSIONS AND EDITING COMMANDS

5.1 Modifiers:

We have seen, in the case of the FOR and the IF modifiers that the execution of a statement can be made to depend upon a modifying clause. I will now give and then describe a list of examples which will include all the modifiers in CAL. The statement numbers are given for reference. This is not a program.

```
1.1 SET A(I)=I FOR I=2 BY 2 TO 14
1.2 SET A(I)=I FOR I=2 BY 2 UNTIL I=14
1.3 SET A(I)=I FOR I=2 BY 2 WHILE I<15
1.4 SET X = Y/(N-1) WHERE N=16
1.5 SET S = SQRT(X) IF X > 0
1.6 SET S = SQRT(X) UNLESS X < 0
```

5.2 For Modifiers with TO, UNTIL and WHILE

Steps 1.1, 1.2, and 1.3 all have the same effect. In each, the "BY" part can be left out when "BY 1" is intended. The terminal conditions in 1.2 and 1.3 (that is the I=14 and I<15) can be quite complicated logical expressions. They need not involve the variable of iteration (I in these examples) directly. A word of caution about WHILE. Many right sounding statements, meaningful in English, can be made which use the word WHILE without using FOR. CAL will not do these statements! Every WHILE must have a FOR part preceding it!

5.3 WHERE Modifies

Step 1.4 illustrates the WHERE modifier. It allows us to make a SET type of statement as a modifier and is a handy way of slipping in a variable, here N, which we had forgotten to define earlier in the program.

5.4 IF and UNLESS Modifiers:

Step 1.5 is the already familiar IF modifier and 1.6 is similar except that the statement is executed only when the condition following the UNLESS is false; thus the two examples will have the same effect except for the case when X is equal to zero.

5.5 Several Modifiers

CAL has no fixed limit to the number of modifiers that can follow a single statement, but the size of the input buffer, which holds the statement as you type it in, does set a limit of 300 characters to the length of a CAL statement. When more than one modifier follows a statement these are interpreted from right to left. For example:

```
1.7 SET A(I)=I FOR I=1 TO N WHERE N=6
```

will SET the first six A(I)s, no matter what the previous value of N was, as it will do the WHERE before the FOR. The use of several modifiers makes it possible to pack quite a lot of program into a single statement. This is not particularly good programming practice, as it makes it harder to understand the program. It is also harder to change. Commas can be used to separate the statement from its modifiers and the modifiers from each other. This may make the statement easier to read. An example of a statement using several modifiers is:

```
2.1 SET A=(A+N/A)/2, FOR I=1 UNTIL ABS (A-N/A)<D, WHERE
D=N/10E7,WHERE A=N/2
```

This will compute the square root of N (by successive approximations using N/2 as the initial estimate of the square root) and call it A.

5.6 Logical Expressions:

So far when we have referred to expressions we have been thinking about arithmetic expressions. CAL also uses logical expressions. CAL considers 0 to be false and any other value to be true. For example 2.3 TYPE A(I), IF A(I), FOR I=1 TO 10 will type only those A(I)s which are not equal to zero.

5.7 Logical Operators

The logical operators are AND, OR, and NOT. By using these operators quite complicated logical expressions can be built up and used as conditions after UNTIL, WHILE, UNLESS, and IF.

If A is false, that is to say, has the value zero then NOT A is true and will be given the value one.

5.8 Relations

Usually logical variables are derived from relations. The six relations used in CAL are =, ≠ (not equal), <, <=, >, and >=. A simple example of how the relations and logical operators can be used is:

```
2.4 SET B = B + A(I), FOR I = 1 UNTIL A(I) = 0 OR I>20, WHERE
B = 0
```

The program will continue to put the partial sum of the A(I)s into B until one or the other of these conditions is true--then it will go on to the next statement. CAL will not complain if you wish to mix logical and arithmetic operations and variables in the same expression so you could have

```
2.5 SET A(I) = B(I) * (NOT (C(I) = D(I)), FOR I = 1 TO 20
```

The effect of this is to SET A(I) = B(I) when C(I) was not equal to D(I) and zero when it was. The logical operators have an order of precedence just as the arithmetic operators do. First the OR and the NOT are done and then the AND. In an expression which contains arithmetic operators, comparisons, and logical operators they are done in that order. One must be careful about this, for example, in the expression A OR B = 4 CAL will first see if B equals four and then OR the result of this with A. Thus it will not be the same as A=4 OR

B=4 even though we often say the first when we mean the second.

5.9 Editing Commands:

In order to use the editing commands of CAL we must know how CAL handles the input from our teletypewriter. CAL has a temporary storage area where the characters from the teletype are stored as they come in. (Such an area is usually called an input buffer by programmers). When CAL receives a (return) it then tries to interpret the string of characters, which came before the (return). If it succeeds it lets us know by typing a >, if not it gives a ? or an error message. In any case the line we have typed stays in the input buffer. If CAL accepted the statement, a copy of it will also be in the program storage area. As we type our next command, we will replace character by character our last input with our new input. The (return) at the end of the new line causes any remaining part of the old line to be thrown away in addition to its effects mentioned above.

All the editing commands are made by holding the control key down while typing the specified command key. We shall continue to indicate this by (c) in front of the command letter. I now give a list of all the editing commands with a brief description of their action.

<u>5.10 DELET commands:</u>

(c)A	This deletes the previous character or space and types a ↑. It can be used several times to delete as many characters.
(c)W	This deletes the preceding word, including the blank in in front of it, and types \.
(c)Q	This deletes everything up to the last line feed and prints ←.

<u>5.11 COPY commands</u>

(c)C	Copies the next character of the old line to the new line and types it.
(c)Z&	This copies up to and including the first time the letter following the (c)Z occurs in the old line. Here I have used & to stand for any character. You do not hold the control key down when you type it.
(c)D	This command copies the rest of the old line into the new one including the (return).
(c)F	This has exactly the same effect as (c) D except that the text copied from the old line to the new one is not typed out on the teletype. Either of these two commands can be used to cause a direct statement to be executed several times.

<u>5.12 SKIP commands</u>

(c)S	Skips over a character in the old line and prints %. This is useful when used with the repeat key.
(c)X&	This acts like (c)Z& except the characters are skipped, including the one typed after the (c)X. It will print a % for every character skipped. Both (c)Z& and (c)X& will ring the bell on the teletype if the character indicated is not found in the old line.

5.13 INSERTIONS

(c)E	Prints and starts inserting whatever is typed next. A second (c)E stops the insertion and types >.

5.14 RETYPING

(c)Y	This types the rest of the old line and then on a new line the new line so far. You can then continue editing the line.
(c)T	This has much the same effect as (c)Y except that it aligns the two lines exactly. These two commands are used when one is confused about just what has been done in the editing of a line.

5.15 EDIT STEP X.Y

This is a CAL statement and it will cause the statement numbered X.Y to be typed out. This statement then becomes the old line for editing purposes. This does not remove it from the program so that if it gets a new statement number in the edit the original step will remain as part of the program as well as the new step generated by the edit.

The following example illustrates the use of these commands.

Suppose that we have a statement

```
1.5 SET U = A + B*X*Y +C*Y↑2
```

which we would like to change so that it read

```
1.5 SET U = A*X↑2 +B*X*Y + C*Y↑2
```

We can type

```
> EDIT STEP 1.5 (return)
1.5 SET U = A*+B*X*Y +C*Y↑2
```

We next type (c)ZA and the computer will type

```
1.5 SET U = A
```

(e)E will make the line look like

```
1.5 SET U = A<
```

we now type *X↑2(c)E to obtain

```
1.5 SET U = A<*X↑2>
```

(c)D will add on the rest of the line.

```
1.5 SET U = A<*X↑2> +B*X*Y +C*Y↑2
```

To check we can type (c)D again to get

```
1.5 SET U = A*X↑2 +B*X*Y +C*Y↑2
```

Note that the use of the (c)E to insert the extra characters kept the rest of the line from being changed so that we could add it on later with the (c)D.

CHAPTER 6: FUNCTIONS

6.1 DEFINING FUNCTIONS

We have already discussed the built-in functions of CAL. We will now explain how a programmer can define his own functions. In most cases a function has a single value, which is determined by one or more arguments, or inputs. For the moment I will discuss only this type of function. A specific example of such a function is:

$$f(x,y) = 3x^2 + 2xy - 5y^2.$$

We define this function in CAL by typing the direct statement

```
DEFINE
F[X,Y] = 3*X↑2 + 2*X*Y - 5*Y↑2.
```

For those without second sight the [is an uppercase K and the] is an uppercase M on the teletype. There is no indication of these brackets on the keyboard.

Now that we have defined a function here are some examples illustrating its use: (In computer lingo these are known as function calls.)

```
>TYPE F[1,3]
      F[1,3] = -36
>TYPE F [34,24]
      F[34,24] = 2220
```

Once a letter has been used to denote a function it cannot be used as a variable name.

When a function is called, as in the first TYPE statement above, CAL saves the values of the variables used as arguments in the definition. In our example these are the variables X and Y. It then sets these variables to have the values of the arguments in the function call. In our example it will SET X=1 and Y=3. It will then find the value of the function according to the definition, and finally it will put the original values of X and Y back as the value of these variables. Thus the variables X and Y used in the definition are protected from being changed when the function is called. The only time this arrangement can cause trouble is when the function is called with the same letters, used as arguments, as were used in the definition but in a different order. Thus in our example F[Y,X] will not have the correct value when X is different from Y.

Occasionally we will wish to define a function that is too

complicated to be defined in one statement. CAL provides a second way of defining functions to allow for this.
An example is:

```
DEFINE G [X,Y] : TO STEP 1.1
1.1 SET B = O
1.2 SET B = B + X ↑ I FOR I = -1 BY -1 UNTIL X ↑ I 0.0001
1.3 RETURN SQRT(B/Y)
```

Note again the DEFINE statement is direct, unnumbered.

When this function is called it will save X and Y and put the values of the arguments into X and Y exactly as outlined above. CAL will then go to the step indicated, STEP 1.1 in our example and continue until, the program comes to the word RETURN. CAL then assigns to the function the value of the expression following the word RETURN. It will then restore the old values to X and Y and continue with the statement which called the function. The statement following the semicolon in the define statement does not have to be a TO STEP statement. If any other statement is used CAL will execute it and RETURN O as the value of the function.

6.2 RECURSIVE FUNCTIONS

CAL functions can be defined recursively. This means that a function can be used as part of its own definition. The following definition of N! (the product of the first N integers) illustrates this clearly, although it is not a very good way to calculate N! in CAL.

```
DEFINE F [N] : TO STEP 2.1
2.1 RETURN 1 IF (N<2)
2.2 RETURN N*F [N-1]
```

To see how this works let us go through the steps needed to evaluate F [3]. Since the argument is not less than two we must evaluate 3*F[2], according to STEP 2.2. To do this we must evaluate F[2]. Here again the argument is not less than two so again we go to STEP 2.2 and try to evaluate 2*F[1]. F[1] has an argument which is less than two and so it RETURNs the value 1, F[2] can now be evalauted, as 2, and finally we see that F[3] = 3*2 or 6 and this will be the value returned by the function. It is clear from this example that CAL must set up quite a lot of machinery to handle this type of function definition so that a direct definition is usally superior. Some functions are much easier to define using recussion, however. The Fibonacci numbers are an example of this.

Fibonacci numbers have been investigated by mathematicians for over 700 years. .They are defined by the rules F(1) = 1,F(2) = 1 and F(n) = F(n-1) + F(n-2), when n>2. The following CAL program computes the first 10 of these numbers.

```
DEFINE F[N] : TO STEP 20.1

20.1 RETURN 1 IF N<3
20.2 RETURN F[N-1]+F[N-2]
>

>TYPE F[I] FOR I = 1 TO 10

  F[I]    =        1
  F[I]    =        1
  F[I]    =        2
  F[I]    =        3
  F[I]    =        5
  F[I]    =        8
  F[I]    =       13
  F[I]    =       21
  F[I]    =       34
  F[I]    =       55
>
```

CHAPTER 7: SAVING PROGRAMS AND DATA ON FILES

The Harvard time sharing system has provision for saving both programs and data from one login to the next. To save a program we use the DUMP command as follows:

```
>DUMP (return)
TO /NAME/ NEW FILE .
```

The name of the file (in this case NAME) can be any set of up to nine characters. Some of these can be non-printing control characters if desired. If you already have a file with the name specified the program will print OLD FILE to remind you that you will lose what was on the file if you put new material into it. DUMP will only save indirect program steps, forms, and function definitions. If does not save variables or direct statements.

To recover a program which has previously been saved on a file named /ABC/ by using the DUMP command we use the following CAL command:

```
>LOAD (return)
FROM /ABC/.
```

Notice that both the LOAD and the DUMP commands need a final period. The name of the file must be enclosed by / marks. Both of these commands are used as direct commands only.

In order to write data on a file we must use three statements as follows:

```
>OPEN /XYZ/ FOR OUTPUT AS FILE 3
>WRITE ON 3 IN FORM 1 : X(I), FOR I = 1 TO 10
>CLOSE 3
```

These commands will write the ten values of X(I) onto a file names XYZ in the format specified by form 1. If the phrase "IN FORM 1" is omitted the numbers will be put on the file in the same format as the TYPE command uses.

To read data from a file we use the processes:

```
>OPEN /XYZ/ FOR INPUT AS FILE 5
>READ FROM 5 : X(I) FOR I = 1 TO 10
>CLOSE 3
```

When CAL reads from a file all alphabetic characters are ignored and only the numeric ones are picked up. If we write an array without using a FORM the value of the index as well as the value of the variable will be on the file. When we ask CAL to read this file it will take the value of the index as the first number and the value of the variable as the second. Thus if we write a file using

```
>WRITE ON 3 : X(I), FOR I = 1 TO 2
```

the record will be

```
X(1) = 17
X(2) = 88
```

and so the record consists of the four numbers 1,17,2, and 88. We can avoid this problem by using a form for our output.

APPENDIX OF CAL STATEMENTS

This appendix lists all the CAL statements known to exist on the Harvard system at the moment. When more are added this list will be updated and distributed. Throughout the discussion e and e(i) denote CAL expressions while v and v(i) stand for CAL variables. Since an expression can consist of a single variable the class of e's includes that of v's but not vice-versa. CAL expressions include logical as well as algebraic expressions and they may involve both built-in functuins and functions defined by the programmer.

<u>Type Statements:</u>

TYPE e_1, e_2, e_3 ----

TYPE IN FORM e: <u>e_1, e_2, e_3, ----</u>

This statement, with a null expression list, can be used to print headings and captions using suitable forms.

TYPE STEP (Statement No.)

TYPE PART e

TYPE ALL STEPS

TYPE ALL FORMS

TYPE ALL VALUES

TYPE ALL

TYPE FORM e

TYPE "immediate string of comment"

This statement can be used to print headings and captions.

<u>Set Statements:</u>

SET v=e

SET v←e

SET v_1←v_2←c

<u>Delete Statements:</u>

DELETE v

DELETE ALL

DELETE FORM e

DELETE PART e

DELETE STEP (Stat. No.)

DELETE ALL VALUES

To Statements:

TO STEP (Stat. No.)

TO PART e

Do Statements:

DO STEP (Stat. No.)

DO PART e

I/O of Data to or from Files, Statements:

OPEN "filename" FOR INPUT AS FILE e

OPEN "filename" FOR OUTPUT AS FILE e

Filename is an arbitrary string of characters not bracketed by slashes or single quotes.

READ FROM e: v_1, v_2, v_3, ----

WRITE ON e_1 IN FORM e_2 : e_3, e , e ----

WRITE ON e_1 : e_3, e , e ...

CLOSE e

To I/O data to and from files is necessary to use three statements:

Open Statement

I/O Statement

Close Statement

<u>Edit Statements</u>:

EDIT STEP (Stat. No.)

EDIT FORM e

<u>Definition of Functions Statements</u>:

DEFINE v[v_1, v_2 ---]=e

DEFINE v[v_2,v_2 ---]: statement   (the statement is usually, TO STEP e)

<u>Miscellaneous Statements</u>:

DEMAND IN FORM e: v_1,v_2, v_3, ---- works only with # type forms

DEMAND v_1, v_2 ...

PAUSE

GO

DUMP (return)

TO /filename/

LOAD (return)

FROM /filename/

PAGE

LINE

CANCEL

DONE

RETURN e

STEP

CAL Modifiers:

Any number of modifiers can be attached to a CAL statement according to the formats:

Statement, Modifier 1, Modifier 2----

Statement Modifier 1 Modifier 2----

WHERE v=e

WHERE v←e

IF e (is non zero)

UNLESS e (is equal to zero)

```
               BY e2          TO e3
                              WHILE e3    is greater than zero
 FOR v=e1      1 if empty     UNTIL e3    as long as e3 is


               BY e2          TO e3
                              WHILE e3
 FOR v←e1      1 if empty     UNTIL e3
```