T H E   U N I V E R S I T Y   O F   M I C H I G A N

Memorandum 36

THE CAMA INTERPRETER

T. J. Dingwall
L. J. Julyk
L. W. Wolf

CONCOMP: Research in Conversational Use of Computers
ORA Project 07449
F. H. Westervelt, Director

supported by:

DEPARTMENT OF DEFENSE
ADVANCED RESEARCH PROJECTS AGENCY
WASHINGTON, D.C.

CONTRACT NO. DA-49-083 OSA-3050
ARPA ORDER NO. 716

administered through:

OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

September 1970

engr

UMR1187

ABSTRACT

The CAMA interpreter allows subroutines to be dynamically loaded and executed, and data to be entered into the CAMA data structure. It also allows modes of data types and sizes of data regions to be specified easily. And, in addition, it allows the user to perform tasks immediately the necessity for which he may not have anticipated. Through its connection with the macro processor the interpreter also allows a convenient and dynamically expandable command languages for use in CAMA.

PREFACE

Since this report was written, we have implemented a more advanced version of the interpreter, superseding the one described here. The new version has a number of additional features and some changes in syntax. The principal features are:

1. The new version is reentrant.
2. Instead of being limited to four modes, an indefinite mode-defining and -supplementing capability has been added.
3. A component structure feature has been added which allows structures up to a depth of five to be referenced.
4. Increased default capabilities which make it easier to specify defaults have been included.
5. A feature has been added which allows the user to specify easily his own subscripting algorithm at each of the five component levels for up to four subscripts per level.

TABLE OF CONTENTS

1. INTRODUCTION

To enhance the capabilities of the CAMA (Computer-aided Mathematical Analysis) system,[1-3] we have written a primitive interpreter. In conjunction with a parser and the macro processor,[4] it permits the creation of flexible and powerful interpretive languages. Also, the interpreter and macro processor together form the command language for CAMA.

The interpreter can do two things: it can call subroutines, and it can enter data into the data structure. Both capabilities require little more than argument-management routines. Hence, the majority of code in the interpreter is concerned with arguments, both constants and variables. To call a subroutine, its name and arguments are given. The arguments are collected into a parameter list, and the subroutine is dynamically loaded and executed.

One of the principal uses of the interpreter is to augment compiled programs. When a compiled program is coded, it is not always possible to anticipate all the results that need to be printed or displayed. Furthermore, some calculations can be performed only after a preliminary examination of the results of the run. When the CAMA data structure is in use, the interpreter can perform side calculations, and display curves and intermediate printed data.

Data entry merely involves the movement of data between arguments.

The interpreter handles several types of constants, including integers and floating point numbers, character strings and hexadecimal data. Processing of constants consists merely of conversion to internal form.

Variables are divided into two classifications according to their lifespan. Temporary variables are expected to be used for a very short time, at the most for the duration of a single CAMA run. Permanent variables, on the other hand, can be saved and restored across runs, along with the rest of the structure.

Variables are also classified according to data type. Presently the available types are arithmetic (scalar), matrix, polynomial, and character string.

In short, through its ability to call subroutines and manipulate arguments, this program provides the most basic capabilities of an interpreter. When augmented with the macro processor and a parser, we arrive at a useful and easily modifiable full-scale interpreter.

2. GLOSSARY

Interpreter

An interpreter is a program which accepts statements in a given language, and immediately performs, statement-by-statement, the actions requested by those statements. This is in contrast to a compiler, which translates the statements of a language as a whole program into an intermediate form for execution at a later time. Interpreters are most often used in an interactive mode with a time-sharing system.

Argument

In the CAMA interpreter, an argument is simply a value. This value can be a constant, either integer or floating-point numeric, a character string, or hexadecimal data; or it can be a variable. A variable can be either a scalar quantity, a matrix, a polynomial, or a character string.

Leading Argument

If a subprogram call statement begins with a scalar variable followed by an equal sign, the subroutine is assumed to be a function type, and the value returned is stored in the scalar variable. This variable is called a leading argument.

Temporary Variable

A temporary variable is a CAMA variable which is not retained for a long period of time. Usually, intermediate results of calculations will be stored in temporary variables.

Any time an 'END' statement is passed to the interpreter, all temporary variables are destroyed. No temporary variables are saved past the end of a CAMA run.

Permanent Variable

A permanent variable is a CAMA variable which contains information that will be needed for a relatively long period of time. Permanent variables can be saved and restored across CAMA runs.

Mode

A mode is a descriptor of a CAMA variable data type. Allowable modes at present are arithmetic (scalar), matrix, polynomial, and character.

Attribute

An attribute gives information about an argument which is not otherwise apparent. An attribute may specify such things as mode, problem, permanent or temporary variable, real or integer, etc.

Problem

A problem is a grouping of related permanent CAMA variables.

Operation of the Interpreter

There are, at present, four modes in the CAMA interpreter. They are

A	arithmetic or scalar mode,
M	matrix mode,
P	polynomial mode,
C	character mode.

The data and operations called by the interpreter vary depending on the mode of variables.

The mode of all temporary variables is set by the user at the beginning of a run, and becomes the default mode for the remainder of the current operation. If the user wishes to process the variable in another mode, he can do so by changing the mode attribute. Permanent variables have their mode set in an association table connected with a given problem name.

Attributes are set by denoting the argument variable or constant followed by an "at" sign (@), followed by one of nine sets of symbols representing the attribute being set. These are

@X	identifies the variable being set as hexadecimal
@R&	identifies the variable being set as 0-byte real
@R4	identifies the variable being set as 4-byte real

@I2	identifies the variable being set as 2-byte integer
@I4	identifies the variable being set as 4-byte integer
@M=mode	changes the mode of a variable from default mode to current operating mode
@P=problem	used when a variable is used from a different problem from the current operating problem
@S	denotes the variable as a system or temporary variable.
@D=(m) or (m,n)	sets the dimension of a polynomial or an array

The first five of these are used only on the left-hand side of a "function type" subprogram call. They allow the data to be stored according to the proper format.

Variables are distinguished by an alphabetic character in the lead position followed by seven or less characters. These can be chosen from the entire repertory of characters, with the exception of those used as delimiters.

The delimiters are:

ƀ	(blank)
@	(at sign)
,	(comma)
;	(semicolon)

and ((left parenthesis)
Right parenthesis is only truncated as a delimiter if it is preceded by a left parenthesis.

Numeric constants are denoted by a numeric character as they are in FORTRAN, with the following slight modifications. An integer without any modifying attribute is considered to be defaulted to a four-byte integer. A floating-point real number without a modifying attribute is defaulted to a four-byte real number. If the user wishes to treat the integers as two-byte integers, he must attach the attribute @I2. Similarly, if the user wishes to store a real constant as a double-precision number, he must attach the attribute @R8.

Variables may be subscripted in the same way that they are subscripted in FORTRAN. At present, this is applicable only in the case of the matrix mode or polynomial mode.

The operation of the interpreter is started by giving the symbols (INTERP). The next line gives the mode, problem, and problem-name, each enclosed in a separate set of parentheses to be used as defaults throughout the remainder of the operations. If the problem name is not given, the problem defaults to SYSTEM, which implies that only temporary variables are being used. If the mode is not given, the mode is defaulted to arithmetic mode.

An interpreter run is terminated by the word END enclosed in parentheses (END). This destroys all temporary (default) data.

Data are entered into variables by means of the data entry statement, which consists of a variable, a double equal-sign, and the data. The data may be a single constant value, a variable, or an array. All of these fields are delimited by a single blank, a multiple blank, a comma, or a comma with or without as many blanks as desired.

Subroutine calls are made by listing the subroutine name and the arguments in the proper calling sequence. Use of the function type subprogram is similar; however, the variable to which the result is to be assigned precedes the name of the subprogram, with a single equal-sign between. All the fields are delimited by blanks and commas as in the data assignment statement.

3. EXAMPLES

These simple examples assume that the following subroutines exist and that they perform the appropriate operations on matrices:

AD	adds two matrices,
SB	subtracts two matrices,
MM	multiplies two matrices,
TR	forms the trace of a matrix,
SC	a scalar times a matrix.

```
(INTERP)
(MAT)(ROUGH)
```

This puts the user in the CAMA interpreter, sets the default mode to matrix mode, and sets the problem name to ROUGH. It implies that the data will be found in the data structure by accessing the list ROUGH. The first left parenthesis must be in column 1. If the mode is not specified, the interpreter will look for the mode information in an association table associated with the problem name. If none is available, error comments will be issued. When the problem name is not specified, the interpreter defaults to SYSTEM or temporary as the storage for the problem. If, under these circumstances, the variable name has not been defined, an error comment is issued.

```
MM A B C
```

This multiplies the matrix A times the matrix B and stores the result in C. Dimension and size information are assumed to be already stored in the matrices. It also assumes proper data in A and B. Note: blank delimiters must be present.

```
Z == 4.3
```

This stores the real*4 number 4.3 into Z as a scalar quantity. Note: blank delimiters must be present. Double-equals is assignment.

```
SC,Z,A,D@S
```

This multiplies the scalar Z times the matrix A and stores the result D in the temporary storage, not in the storage under the problem-name ROUGH. Note that commas can replace blanks as delimiters.

```
A@D=(3,4) == 3.5,2.3,0.
```

This sets the dimensions of A to a 3x4 matrix and stores the succeeding values beginning with A(1,1) by rows according to the CAMA matrix format. Note commas and blanks are used as delimiters.

```
V(3,2) == V(3,3)
```

This sets the matrix element V(3,2) to the same value as V(3,3).

```
AD A B C;SB C Q R;AD R X Y
```

This adds the matrix A to B and puts result in C. Then it subtracts Q from C and stores result in R. Then it adds X to R and stores it in Y. Note the use of a semicolon to delimit more than one statement on a line.

```
(POLY)
```

This changes the mode to polynomial mode, but leaves the problem name the same, ROUGH.

```
( )(EASY)
```

This leaves the mode the same but changes the problem name.

```
(MAT)(ROUGH)
```

This changes both the mode and the problem-name back to their original designations.

```
R@M=P == 6.4,9.1,-13.6,1.4E-01
```

The values on the right-hand side of the assignment statement are stored and treated as a polynomial, not as a matrix. R must have been specified as a polynomial in some previous run.

```
GALOP@P=HARD == 6.17
```

The value 6.17 is stored in GALOP(1,1) in problem HARD.

```
MM A,GALLOP@P=HARD,M@S
```

The matrix A of the problem ROUGH is multiplied by GALLOP of the problem HARD. The result is stored in the temporary data structure in the matrix M.

```
A*G@R4 = TR A
```

The trace of the matrix A is stored in the variable A*G. Note the * is not an arithmetic operator but a character in the variable name.

```
A1@P=JOE@M=A    ==,-1.E+4
```

This sets the real*4 number in problem JOE, whose mode is scalar, to the value 10,000. Note: the multiple blanks and comma are alternate delimiters.

```
BOY@R8 == 1.763479189
```

This sets the double-precision variable BOY to 1.763479189.

```
CAT@I2 == 6
```

CAT is set to the two-byte integer value 6.

Example 1. Storing and Printing a Polynomial

```
(INT)(POLYNOMIAL)\           This sets the default mode of
                             temporary variables to poly-
TK 6530                      nomial

POLY == 4.0 3.0 2.0 1.0\     The polynomial 'POLY' in tempor-
                             ary storage is created and given
TK 6530                      values

LIMIT 0.0 1.0 0.1\           LIMITS are set for polynomial
                             evaluation
P POLY                       The polynomial is evaluated

     POLY      POLY      POLY      POLY      POLY      POLY

POLY( 0)= 0.4000000E 01    X=   0.0     VALUE= 0.4000000E 01
POLY( 1)= 0.3000000E 01    X=   0.100   VALUE= 0.4320999E 01
POLY( 2)= 0.2000000E 01    X=   0.200   VALUE= 0.4687999E 01
POLY( 3)= 0.1000000E 01    X=   0.300   VALUE= 0.5106998E 01
                           X=   0.400   VALUE= 0.5583999E 01
                           X=   0.500   VALUE= 0.6124998E 01
                           X=   0.600   VALUE= 0.6735997E 01
                           X=   0.700   VALUE= 0.7422997E 01
                           X=   0.800   VALUE= 0.8191997E 01
                           X=   0.900   VALUE= 0.9048997E 01
                           X=   1.000   VALUE= 0.9999994E 01


                           X=   1.000   VALUE= 0.1000000E 02

     POLY      POLY      POLY      POLY      POLY      POLY

TK 6530
```

The reverse slash (\) is generated by CAMA as an indication that the line has been received. TK6530 is returned by CAMA to indicate that it is prepared to receive a new line.

Example 2. Use of Character-String Mode

```
(INT)(CHARACTER)\                           Default mode is set
                                            to character
TK 6530

STRING == THIS IS A CHARACTER STRING\       Character variable
                                            is created and set
TK 6530

SPRINT STRING 26@I2 0 0\                    Character string
                                            used as argument
 THIS IS A CHARACTER STRING

TK 6530
```

Example 3. Use of Function-Type Subprograms and Hexa-
decimal Constant

```
(INT)(ARITHMETIC)\   default mode set to arithmetic

TK 6530

MPTR@I4 = MASPTR\    master directory pointer fetched
                     and stored in MPTR
TK 6530

FN MPTR 6F000000@X 6F000000@X\  master directory pointer
                                used for dump.  Note use
                                of hexadecimal constants.
     DUMP OF MASDIR
     FILELN    00502500
     MASASSO   00517A70
     ATPL      005091C0
     BUFFERPL  00514FD8         6F is hexadecimal for (?)
     COMTAB    00502B40         The two question marks
     DFREADPL  00509FF0         say dump everything in
     LPARPACK  00516F88         the list.
     MLANGDIR  00501E98         Whenever the address
     MPROB     005013B8         portion is null (000000)
     MSYMTBL   00502FD0         the contents are printed
     NAMEFLAG  00509138         on the output device.
     PAUSEPL   0050AFE0
     PROBLEM   00509170
     PROBLIST  00516D20
     QUE/LIST  005091Q0
     READPL    00591E8
     SSG       0051B020
TK 6530
```

The hexadecimal constants 6F000000 are equivalent to a question-mark left-justified, padded with zeros. When given as the second and third arguments dump the whole master list.

Example 4. Use of Permanent and Temporary Storage
under Different Problem Names

```
(INT)(MATRIX)(PROB1)\  default mode set to matrix; de-
                       fault problem set to  PROB1
TK 6530

PM MAT1@S@D=(2,2)\     temporary 2x2 matrix created,
                       zeroed, and used in call to
                       print routine.
                       If the dimension were not specified
                       MAT1 would have been created 25x25
                       zeroed and printed.

       MAT 1      2 ROWS       2 COLUMNS      Page 1

             COLUMN      1               2

           ROW   1   0.0             0 0
           ROW   2   0.0             0.0
TK 6530
MAT1@S == 2.0 3.0 1.0 5.0; PM MAT1@S\    MAT1 given values
                                         and printed.


       MAT 1      2 ROWS       2 COLUMNS      PAGE 1

             COLUMN      1               2

           ROW   1   0.200000E 01    0.300000E 01
           ROW   2   0.100000E 01    0.500000E 01
TK 6530
MAT2 == 7.0 1.5 3.7 -8.2\    permanent matrix  MAT2
TK 6530                      given values
TK 6530
MAT2(2,1) == 6.0 ; PM MAT2\  an individual element of
                              MAT2  given a value.  MAT2
                             is printed.  Matrices are
                             stored by rows.  Dimension
                             information for MAT2 had
                             been previously specified.

       MAT2      2 ROWS       2 COLUMNS      PAGE 1

             COLUMN      1               2

           ROW   1   0.700000E 01    0.150000E 01
           ROW   2   0.600000E 01   -0.820000E 01
TK 6530
```

```
MA MAT1@S MAT2 MAT3@P=PROB2\  MAT1 and MAT2 are added
                              and the result stored in
TK 6530                       MAT3, a permanent variable
                              in PROB2.
PM MAT3@P=PROB2\              MAT3 is printed out.


       MAT3        2 ROWS       2 COLUMNS       PAGE 1

             COLUMN       1                 2

          ROW   1   0.900000E 01      0.450000E 01
          ROW   2   0.700000E 01     -0.320000E 01
TK 6530
(INT)(END)\
TK 6530                       END statement — all tempor-
                              ary variables destroyed,
                              and all defaults cleared.
%MTS\                         Control is returned to MTS.
TK 1375 0
# $5.73
```

REFERENCES

1. Julyk, L.J., The CAMA Operating System, Memorandum 30, Concomp Project, University of Michigan, Ann Arbor, August 1970.

2. Julyk, L.J., The CAMA Data Structure, Memorandum 29, ibid.

3. Wolf, L.W., CAMA (Computer-Aided Mathematical Analysis): A General Description, Memorandum 33, ibid.

4. Dingwall, T., Julyk, L.J., and Wolf, L.W., The CAMA Macro Processor, Memorandum 35, ibid.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) UNIVERSITY OF MICHIGAN CONCOMP PROJECT	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE

The CAMA Interpreter

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Memorandum

5. AUTHOR(S) (First name, middle initial, last name)

T.J. Dingwall, L.J. Julyk, and L.W. Wolf

6. REPORT DATE September 1970	7a. TOTAL NO. OF PAGES 18	7b. NO. OF REFS 4
8a. CONTRACT OR GRANT NO. DA-49-083 OSA-3050	9a. ORIGINATOR'S REPORT NUMBER(S) Memorandum 36	
b. PROJECT NO.		
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d.		

10. DISTRIBUTION STATEMENT

Qualified requesters may obtain copies of this report from DDC.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency

13. ABSTRACT

The CAMA interpreter allows subroutines to be dynamically loaded and executed, and data to be entered into the CAMA data structure. It also allows modes of data types and sizes of data regions to be specified easily. And, in addition, it allows the user to perform tasks immediately the necessity for which he may not have anticipated. Through its connection with the macro processor the interpreter also allows a convenient and dynamically expandable command language for use in CAMA.

DD FORM 1 NOV 65 1473

Unclassified
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
CAMA interpreter interpreter						