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

Memorandum 35

THE CAMA MACRO PROCESSOR

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

August 1970

TABLE OF CONTENTS

1. INTRODUCTION

Under the CAMA (Computer-Aided Mathematical Analysis) System[1-4], we have devised a special macro processor which has as its object-languages not assembler languages, but higher-order languages such as FORTRAN, MAD, or ALGOL. The macro processor was designed to accomplish a number of objectives. First, it would enable a relatively unsophisticated user who is acquainted only with a language such as FORTRAN and not with the assembly language, to create macros and bases of languages without having to code in the assembly language. Second, the macro language was created to be a preprocessor for the interpreter language in CAMA. Third, it was created to be an intermediate processor between the mathematical expressions generated in the terminal computer which pass through a parsing operation and the base language such as FORTRAN. The macro processor was also created so that commands for the CAMA system could be written and extended easily, thereby enabling relatively unsophisticated users to extend the commands for their own particular needs, as well as write the original system commands.

The user may create his own languages by means of certain operational macros which, when expanded, generate FORTRAN statements. Or he may create languages which generate statements in a language he or someone else has created. These statements eventually will be expanded into the base language such as FORTRAN or MAD. They may also go through

the interpreter. The word "language" in this context means that in defining the macro, the user has created a certain pattern whose meaning is defined by the macros that make up that language. There is very little language structure in the form that is normally known as syntax, except for that already specified in the macro processor. For example, the operation for adding two matrices is quite different from that of adding two scalar numbers. Yet in both a matrix language and a scalar language, the word "add" could be used to specify the addition process. The output of the macro processor can be processed either through the CAMA interpreter or through a compiler for the language which has been specified as the base language for all of the macros. However, the base language for all macros must be the same. That is, we cannot expand some macros into FORTRAN and others into MAD and process them all through the same compiler. The user must observe this rule because the macro processor does not check, unless notified, to see whether the base language is self-consistent.

The CAMA macro processor was modeled on the macro processor for the IBM model 360 assembler, and although it is not an exact replica it was created in the same spirit, and uses some of the same notation, terminology, and operational procedures[5]. The peculiarities of the CAMA system will be explained in succeeding parts of this report.

2. GLOSSARY

Symbolic Parameter

A symbolic parameter consists of no more than eight alphanumeric characters preceded by an ampersand. The eight alphanumeric characters may begin with an alphabetic or a numeric character, may consist of entirely alphabetic or entirely numeric characters, or a mixture of them, and may also include a number of special symbols such as the question mark. However, a number of symbols are excluded (see the definition, Excluded Symbols). The symbolic parameters are used in the macro definition as arguments in the prototype statement, and as the dummy symbols in the model statements. That is, any time the macro is called, these symbolic parameters are substituted for in the model statements according to the way they have been called in the call statement and according to the format of the prototype. The substitution is a character-string-type substitution (see Examples).

Excluded Symbols

ø , ( ) ' & @ + - * / . =

Interpreter

An interpreter is the program that accepts statements in a certain simplified format, and processes and executes them immeidately. It is to be used with time-sharing or

real-time processing, and allows the user to get immediate results. This is in contrast to a compiler which compiles the complete procedure and then waits for the user to call that procedure before executing it. The interpreter depends upon the existence of subroutines which, in effect, are called by the interpreter. These subroutines do the processing for the user. For example, to add two matrices, the user specifies an add command to the interpreter, and the two matrices (provided they are predefined) are added immediately.

Macro Processor

A macro processor takes certain lines or strings of characters and substitutes other strings or characters for them. Not all of the string is substituted, and the output string is presumably in the form that a language processor can handle. For example, the output string may be in the form of an input string to the FORTRAN compiler. Certain variable names in that FORTRAN string may not be the same every time the string is to be used. Therefore, to put the string in the proper format, the macro processor substitutes user-specified names into the string each time macro is called (see Example 1).

Attributes

An attribute is a characteristic of a variable or an

argument in the macro processor. As far as the macro processor is concerned, there are only two types of attributes: status attributes and count attributes. Details of the operations and functions of these two attributes can be found in the section explaining their operation.

Local Set Symbol

A local set symbol is a symbolic parameter defined and used within a single macro. Its value may be changed within the macro by use of the SETA and SETC commands.

Global Set Symbol

A global set symbol is a parameter which is predefined in an early macro and can be used in any subsequent macro. It contrasts to a local set symbol, which can be used only in the macro in which it is defined.

Prototype Statement

A prototype statement is a first line of a macro definition. It sets the format for that definition and defines the symbolic parameters that will be used as arguments; it also specifies the name of the macro.

Language

In CAMA, the word "language" is used to refer to groups of macros, as well as to such better-known languages as FORTRAN and MAD. These macros can be accessed as a group,

and calls within them are intended specifically for that language or for that set of macros. The same calls may be used in a different set of macros or for a different language under a different language name and in such cases mean different operations. Also, macros that have the same names but under different language names may perform different operations. For example, a group of macros under the language name MATRIX perform matrix operations. There is a different group under the name DOUBLE POLYNOMIAL which does double polynomial operations. Even though within these macro sets or within each of these languages there would be an operation called multiply, the actual operations would be quite different. The word "language" in this context has a more restricted meaning that it does in, say, the context of FORTRAN or other languages. The matrix language or the double-polynomial language, used in connection with the parser, results in a more complete language, more like a full-blown computer language. In this case, the parser actually converts a more complete version of these languages to their macro form.

Keyword Parameter

Keyword parameters are arguments in a prototype statement in a macro definition. They consist of a leading ampersand followed by up to 8 alphanumeric characters, as do symbolic parameters. However, keyword parameters have,

in addition, an equal sign and the default value of that argument following. The default may be null or it may be an empty string of up to 256 characters, including any symbol in the symbol set. If a blank or a comma (normally excluded symbols) are enclosed in parentheses, they will be accepted; similarly, a blank or a comma enclosed between primes will be accepted. If a keyword parameter is not explicitly specified in the call then the information on the right-hand side of the equal sign is substituted in the model statement at the time of macro expansion. Unlike positional parameters, keyword parameters may be given in any order. However, they could also be given as positional parameters; that is, given in the proper position they will be taken for that substitution of that parameter.

Positional Parameter

Positional parameters are symbolic parameters of the macro prototype statement. Substitution, at expansion time, is dependent upon the position in the prototype. The first argument in the calling statement is substituted for the first symbolic parameter, the second argument for the second symbolic parameter and so forth. They differ from keyword parameters only by the fact that no default values are prescribed.

Macro Call

A macro call is a reference to a predefined macro

whose name is referenced in the construction of a procedure or another macro.

String

A string is a sequence of characters--alphabetic, numeric, and/or special--which can be accepted by a digital computer.

Null String (or Null)

A null string is one with no characters and hence occupies no storage. By contrast, a blank string is one which has a blank character.

Default Value

The default value is the string that is substituted during macro expansion when the user does not specify a string. The default value is defined by the keyword parameters in the macro prototype statement.

Leading Symbolic Parameter

In the macro prototype statement, the symbolic parameter or the keyword parameter that precedes the name of the macro is called the leading symbolic parameter. This symbolic parameter is frequently used for statement numbers in FORTRAN or statement labels in other languages.

Macro Command

A macro command is a command to the macro language processor which is executed at the time of macro expansion. Macro commands are control statements which govern the order and the nature of the processing. They are also used for the declaration and manipulation of set symbols, as well as to insert notes and comments to either the writer or the user, particularly if he performs something erroneously.

Concatenation

When two strings are joined so that the second string merely becomes a continuation of the first, they are said to be concatenated. They then may be treated as a single string.

Substring

A substring is part of a string or a subset of the symbols in a larger string. A substring, in general, is a string of adjacent symbols that have been taken from the string.

Arithmetic Symbols

The terms "arithmetic symbol" and "arithmetic value" may be used interchangeably. In certain circumstances in the use of the SETA command, combinations of numbers and

arithmetic operators (plus, minus, multiply, and divide) will be treated not as a string but as the number value that string represents. Also, certain set symbols that have arithmetic values can be either concatenated or used in simple arithmetic expressions to represent a new arithmetic value. See Example 4.

Macro Definition

A macro definition consists of a macro prototype statement, followed by one or more macro model statements, and/or commands, and terminated by a MEND enclosed in parentheses.

Macro Prototype

A line of symbols consisting of a leading symbolic parameter (optional), a macro name, and zero to twenty symbolic parameters.

Macro Model Statement

A line of symbols which is to be reproduced when a macro is expanded. At expansion the symbolic parameters and set symbols occurring in the model statement will have their current values--strings of symbols--substituted.

Operation of the Macro Processor

The macro processor in CAMA is initiated by the enclosing of the word MAC in parentheses, followed by the

macro name enclosed in parentheses, followed by the language's name enclosed by parentheses. Each line of the macro definition must be preceded by a line number. The line number is not shared as part of the text, but can be used for editing purposes, just as are MTS line files. The macro definition is terminated by a MEND statement enclosed in parentheses.

3. EXAMPLES

3.1 SIMPLE MACRO EXAMPLES

Example 1. Prototype

```
  17.3        &ALPHA          GEORGE        &B,&CAT1
    ↑           ↑               ↑             ↑
line          leading         macro         symbolic
number        symbolic        name          parameters
(not stored   parameter                     (from 0 to 20 may
with text)    (optional)                    be specified)
```

Many blanks, one blank, or a single comma accompanied by zero, one, or more blanks can be used as delimiters.

Example 2. Model Statements in

a) a Base Language

```
          16.1(FORTRAN)13      A=B+&C
            ↑        ↑           ↑
line number       language    FORTRAN statement con-
(peeled off       name        taining symbolic para-
and not stored)               meters
```

During expansion of a FORTRAN-based macro, column numbers are aligned to standard FORTRAN.

b) a Macro-created Language

```
      32.2(LANG1)ALPHA          GEORGE      A,ERIC
           ↑       ↑              ↑           ↑
LANG1 is the    leading        macro       macro
language in     argument       name        arguments
which the
macro GEORGE
is to be found.

       9.1(LANG2)                HENRY     A1,&B7
            ↑          ↑          ↑           ↑
LANG2 is the       leading    macro       arguments con-
language in        argument   name        taining symbolic
which the          omitted                parameters
macro HENRY
is to be found.
```

This assumes that LANG1 and LANG2 have been previously created.

Example 3. Simple Usage

The following is a macro definition.

```
                          CAMA control statements
(MACRO)(F6)(TREE)         F6 => macro name; TREE => language
1  &A  F6  &B  &C         prototype statement
2  &A ARE &B AND &C       model statement
(MEND)                    CAMA control statement
```

The macro call

```
(TREE)GIRLS F6 SUGAR SPICE
```

would produce on expansion

GIRLS ARE SUGAR AND SPICE.

Example 4. Cascading of Languages

Macros defined in one language may be used to define new macros, either in that language or another.

Suppose that in the language LANG1 the following macro is defined.

```
  &LEAD  ALFRED  &DOG,&CAT                    prototype
 .(FORTRAN)  &LEAD  ANSWER=(&DOG)*&CAT        model
```

Then this is used in defining a macro in the LANG2 language.

```
&A HENRY &B,&C
(LANG1)&A ALFRED &B+&C,53
```

Then the call

```
(LANG2)691  HENRY  A1  B7
```

would produce on expansion

```
691  ANSWER=(A1+B7)*53
```

in the proper FORTRAN column format. The call

```
(LANG2)  HENRY  MAY-JUNE  SEPT
```

would produce on expansion

```
ANSWER=(MAY-JUNE+SEPT)*53.
```

Example 5. Too Many and Too Few Arguments on Calling a Macro

Given the following prototype

```
MACNM  &A  &B  &C
```

the call

```
MACNM  ALPHA  BETA,
```

would set &A to ALPHA, &B to BETA, and &C null.
Whereas the call

```
MACNM  ALPHA  BETA  GAMMA  OMEGA
```

would set &A to ALPHA, &B to BETA, and &C to GAMMA;
OMEGA is ignored. OMEGA could be referenced as &SYSLIST(4).
If the mode statement for this macro were

```
&A+&B-&C
```

the two expansions would be

```
ALPHA+BETA-
```

and

```
ALPHA+BETA-GAMMA
```

respectively.

Example 6. Use of Keyword Parameters

If &A=XYZ occurs in a prototype, and A=123 occurs

in a macro call, the &A is given the value of 123; otherwise &A retains the value XYZ.

If the literal string X=Z were to be a positional argument, the user would write X==Y in the macro call. If it were used as a keyword argument, A=X=Y would be written.

Example 7. Mixed Keyword and Positional Prototype

If the following was a prototype

```
&LEAD  MACRO  &A=XYZ,&B
```

and the macro call

```
MACRO  123,B=ABC,
```

at expansion time, the macro call sets &A to 123 and &B to ABC.

Example 8. Special Symbols

The symbols ⌀ , ( ) ' & @ . + - * / = are not allowed in a symbolic parameter. To enter an & (ampersand), @ (at sign), or a . (period) in a string being expanded, the user must use a double symbol. The same applies to primes needed between primes in a literal string. If &A was set to BOY and &B set to GIRL, the string &A && &B would be expanded as

BOY & GIRL.

Example 9. The MGO Command

The MGO command transfers operation of the macro processor to the line specified,

(MGO) 6 - reads next model statement from line 6.

Assume &A is 6, &B is 3, and &A5 is 12.4.

```
(MGO)   &A       processing continues at line 6
(MGO)   &A.5     processing continues at line 65
(MGO)   &A..5    processing continues at line 6.5
(MGO)   &A5      processing continues at line 12.4
(MGO)   &A&B     processing continues at line 63
(MGO)   &A.&B    processing continues at line 63
(MGO)   6.5      processing continues at line 6.5
```

Example 10. Use of LOCAL, GLOBAL, SETA, and SETC Commands

```
(LOCAL)  &A, &A69B4
```

declares symbols &A and &A69B4 to be local set symbols (either arithmetic or character).

```
(GLOBAL)  &A, &CAT, &BOWWOW
```

declares symbols to be global set symbols.

```
(SETA) &A = 6
```

sets local or global set symbol &A to 6.

```
(SETA)  &CAT = 6*7
```

sets &CAT to 42.

```
(SETA)  &CAT = &A + &MOUSE
```

If &A is 2 and &MOUSE is 7, &CAT is set to 9.

```
(SETA) &A = &B&C
```

illegal (no concatenation is allowed).

```
(SETC)  &A = 'GOAT'
```

gives &A string value GOAT.

```
     (SETC)  &A = GOAT
```

illegal (primes needed).

```
     (SETC)  &A = 'CAN''T'
```

sets &A to CAN'T.

If &A is DOG, &B is BOWWOW, &M=3, and &N=4

```
     (SETC)  &CAT =  '&A' sets &CAT to DOG
     (SETC)  &CAT =  '&A&B' sets &AAT to DOGBOWWOW
     (SETC)  &CAT =  '&A.B' sets &CAT to DOGB
     (SETC)  &CAT =  'B&A' sets &CAT to BDOG
     (SETC)  &CAT =  '&B'(2,5) sets &CAT to OWWO (picks
                     second through fifth characters in
                     string)
     (SETC)  &CAT = '&B'(&M,&N) uses values of &M and &N
                     to truncate &B to WW
     (SETC)  &CAT =  '&B'(3,10) sets &CAT to WWOWbbbbb
     (SETC)  &CAT =  '&A(2) sets &CAT to OG
     (SETC)  &CAT =  '&A'(,2) sets &CAT to DO.
```

Example 11. Use of the Arithmetic Conditioned Branch MIFA

```
(MIFA) &A EQ 1 25
```

If &A has the value 1, read the next model statement from line 25; otherwise continue with the next statement.

```
(MIFA) @K&R GT 5 &A
```

If &R contains more than five characters and &A contains a valid line number, a branch is made to the

given line. Valid relationals are CT, EQ, LT, GE, NE, and LE.

Example 12. Use of the Character Conditional Branch MIFC

(MIFC) '&A' EQ 'STRING' 75

If &A is the string 'STRING', then a transfer is made to line 75. If strings are unequal in length, the shorter is left-justified and padded with blanks.

Example 13. Use of MNOTE

The command MNOTE is used in the following way. Given in the macro model statement

(MNOTE) '&A' MADE IN PROCESSING XYZ

at expansion time the line

ERROR MADE IN PROCESSING XYZ

will be generated.

Example 14. Use of MCOM Command

The command

(MCOM) THIS IS AN INTERNAL COMMENT

will generate nothing on expansion and is used only for internal reference.

Example 15. Terminal Commands

Any of the commands

(MEXIT),

(MEND), or

(PEND)

terminates macro expansion.

Example 16. Use of @S Attribute

In the model statement

```
(MIFC) '@S&ARG' EQ 'O' 23
```

if @ARG is omitted (null) from the macro call, 23 is the next line processed; otherwise the line following is processed.

When the model statement is

```
(MIFC) '&S&DOG' NE 'A' 47
```

if &DOG is not arithmetic, processing is transferred to line 47.

Example 17. Use of &SYSNDX

If the model statement were

```
(FORTRAN) 2&SYSNDX X=2+2
```

and if this statement occurs in the 25th macro processed,

```
225  X = 2 + 2
```

is generated, thus creating a unique statement number.

Example 18. Use of &SYSLIST

Given the model statement

```
(MIFA)  &SYSLIST GT 25 40
```

if there are more than 25 arguments, go to line 40 for the model statement, otherwise continue with the next line.

```
(SETC) &BETA = '&SYSLIST(5)'
```

&BETA is set to the fifth argument given.

Likewise the model statement is

```
(MIFC) '@S&SYSLIST(0)' EQ 'O' 73
```

if the leading argument is omitted, and control is transferred to line 73.

Example 19. Use of &SYSRTNCD and &SYSRESLT

&SYSRTCD has the contents of general register 15 in it. Likewise &SYSRESLT has the contents of general register zero. The model statements are

```
(INTERP) MASPTR
(MIFA) &SYSRTNCD NE 0 57
(INTERP) FN &SYSRESLT 6F000000@@X 6F000000@@X
```

The second statement transfers control to line 57 if the call on MASPTR has a non-zero return code. The third statement provides a dump of the master directory using the pointer returned by MASPTR.

Example 20. Use of &SYSLEVEL

There is no checking in the macro processor to see if user has generated an infinite recursive loop. The user may test for this himself by use of &SYSLEVEL.

The model statement

```
(MIFA) &SYSLEVEL GT 15 82
```

transfers control to line 82 if macro expansion has gone deeper than 15 levels.

Example 21. Use of Count Attribute @K.

The model statement

(MIFA) @K&DOG GT 8 73

transfers control to statement 73 if &DOG contains more than 8 characters.

3.2 COMPLETE EXAMPLES

Example 1. Simple Substitution

This example shows the simple substitution properties of macros. The arguments passed are substituted to generate valid FORTRAN statements. In the second call, the leading argument is null, and no statement label is generated.

Entry of Macro Definition

```
(MAC)(ADDSUB)(EXAMPLE)\
1&LABEL ADDSUB &A,&B,&C,&D\
2(FORTRAN)&LABEL &C=&A+&B\
3(FORTRAN) &D=&A-&B\
4(MEND)\
```

Listing of Macro Definition

```
%LIST ADDSUB EXAMPLE\
       1     &LABEL ADDSUB &A,&B,&C,&D
       2     (FORTRAN)&LABEL &C=&A+&B
       3     (FORTRAN) &D=&A-&B
       4     (MEND)
```

Entry of Procedure Definition Containing Macro Call

```
(PRO)(EX1)\
1(EXAMPLE)1 ADDSUB X,Y,Z,W\
2 ADDSUB J,K,L,M\
```

Listing of Procedure

```
%LIST EX1
       1      (EXAMPLE)1 ADDSUB X,Y,Z,W
       2      ADDSUB J,K,L,M
```

Expansion of Procedure and Macro

```
%EXPRO EX1
 1      Z=X+Y
        W=X-Y
        L=J+K
        M=J-K
```

Example 2. Use of Keyword Parameters

This example shows the use of keywords. In the first call, all arguments are specified. In the second call, all keywords are allowed to default. In the third call, the second argument, LEN, is given as a positional argument even though it was defined as a keyword parameter.

Note also in the FORTRAN statement, two ampersands were used to generate a single &.

Generating the Macro

```
1 SPRINT &STRING,&LEN=256,&MOD=0,&LINE=0
2(FORTRAN) CALL SPRINT(&STRING,&LEN,&MOD,&LINE,&&10)
3(MEND)
```

Listing the Macro

```
%LIST SPRINT EXAMPLE
       1       SPRINT &STRING,&LEN=256,&MOD=0,&LINE=0
       2       (FORTRAN) CALL SPRINT(&STRING,&LEN,&MOD,
               &LINE,&&10)
       3       (MEND)
```

Generating Macro Calls

```
1(EXAMPLE) SPRINT STRING,LEN=32,MOD=1024,LINE=10
2 SPRINT AREA
3 SPRINT OUTPUT,45,MOD=4096
```

Listing Macro Calls

```
%LIST EX1
       1     (EXAMPLE) SPRINT STRING,LEN=32,MOD=1024,LINE=10
       2     SPRINT AREA
       3     SPRINT OUTPUT,45,MOD=4096
```

Expansion of Procedure

```
%EXPRO EX1
       CALL SPRINT(STRING,32,1024,10,&10)
       CALL SPRINT(AREA,256,0,0,&10)
       CALL SPRINT(OUTPUT,45,4096,0,&10)
```

Example 3. Use of &SYSNDX

This macro demonstrates the use of &SYSNDX to generate unique statement labels. If &SYSNDX were not used, the two calls on VECTADD would generate duplicate statement numbers.

Generation of Macro

```
(MAC)(VECTADD)(EXAMPLE)
1&LABEL VECTADD &A,&B,&C,&DIM
2(FORTRAN)&LABEL DO 10&SYSNDX I=1,&DIM
3(FORTRAN)10&SYSNDX &C(I)=&A(I)+&B(I)
4(MEND)
```

Listing of Macro

```
%LIST VECTADD EXAMPLE
       1      &LABEL VECTADD &A,&B,&C,&DIM
       2      (FORTRAN)&LABEL DO 10&SYSNDX I=1,&DIM
       3      (FORTRAN)10&SYSNDX &C(I)=&A(I)+&B(I)
       4      (MEND)
```

Generation of Procedure

```
(PRO)(EX3)
1(EXAMPLE) VECTADD X,Y,Z,10
2(EXAMPLE)23 VECTADD L,M,N,25
```

Listing of Procedure

```
%LIST EX3
       1      (EXAMPLE) VECTADD X,Y,Z,10
       2      (EXAMPLE) 23 VECTADD L,M,N,25
```

Expansion of Procedure

```
%EXPRO EX3
       DO 102 I=1,10
102    Z(I)=X(I)+Y(I)
23     DO 103 I=1,25
103    N(I)=L(I)+M(I)
```

Example 4. Arithmetic Symbols and Branching

In this example, arithmetic symbols and branching are used. Line 2 defines a local set symbol, &COUNT, which will be used as an index in a loop. The set symbol is also initialized to zero.

Line 3 is the first statement of the loop. This command increments the index by 1. Next, line 4 checks the condition for ending the loop. If the index is greater than the number of arguments passed (&SYSLIST), the macro is finished.

Line 5 generates the actual code. A call on the non-existent subroutine, SUBR, is made using the argument specified by the index &COUNT. Line 6 merely closes the loop, and line 7 terminates the macro.

```
(MAC)(LOOP)(EXAMPLE)
1 LOOP
2(LOCAL) &COUNT
3(SETA) &COUNT=&COUNT+1
4(MIFA) &COUNT GT &SYSLIST 7
5(FORTRAN) CALL SUBR(&SYSLIST(&COUNT))
6(MGO) 3
7(MEND)

%LIST LOOP EXAMPLE/
       1       LOOP
       2      (LOCAL) &COUNT
       3      (SETA) &COUNT=&COUNT+1
       4      (MIFA) &COUNT GT &SYSLIST 7
       5      (FORTRAN) CALL SUBR(&SYSLIST(&COUNT))
       6      (MGO) 3
       7      (MEND)

(PRO)(EX4)
1(EXAMPLE) LOOP A,B,C,D
2(EXAMPLE) LOOP J,K,L,M,N,O
3(EXAMPLE) LOOP X
4(EXAMPLE)

%LIST EX4
       1      (EXAMPLE) LOOP A,B,C,D
       2      (EXAMPLE) LOOP J,K,L,M,N,O
       3      (EXAMPLE) LOOP X
       4      (EXAMPLE) LOOP

%EXPRO EX4
       CALL SUBR(A)
       CALL SUBR(B)
       CALL SUBR(C)
       CALL SUBR(D)
       CALL SUBR(J)
       CALL SUBR(K)
       CALL SUBR(L)
       CALL SUBR(M)
       CALL SUBR(N)
       CALL SUBR(O)
       CALL SUBR(X)
```

Example 5. Use of Global Set Symbols

This macro demonstrates the use of global set symbols. The macro INIT concatenates its two arguments, truncates the string at 8 characters, and sets &STRING

to this value. Later, the macro CALL utilizes the global symbol &STRING.

Generation of Macro INIT

```
(MAC)(INIT)(EXAMPLE)
1 INIT &A,&B
2(GLOBAL) &STRING
3(SETC) &STRING='&A&B'(1,8)
4(MEND)
```

Generation of Macro CALL

```
(MAC)(CALL)(EXAMPLE)
1 CALL
2(GLOBAL) &STRING
3(FORTRAN) PRINT 101
4(FORTRAN)101 FORMAT('&STRING')
```

Listing of Macro INIT

```
%LIST INIT EXAMPLE
     1         INIT &A,&B
     2        (GLOBAL) &STRING
     3        (SETC) &STRING='&A&B'(1,8)
     4        (MEND)
```

Listing of Macro CALL

```
%LIST CALL EXAMPLE
     1         CALL
     2        (GLOBAL) &STRING
     3        (FORTRAN) PRINT 101
     4        (FORTRAN) 101 FORMAT('&STRING')
```

Generation of Procedure

```
(PRO)(EX5)
1(EXAMPLE) INIT ABC,DEF
2 CALL
3(EXAMPLE) INIT 12345,67890
4 CALL
```

Listing of Procedure

```
%LIST EX5
       1      (EXAMPLE) INIT ABC,DEF
       2       CALL
       3      (EXAMPLE) INIT 12345,67890
       4       CALL
```

Note: The use of (EXAMPLE) in line 3 is unnecessary. It could have been deleted so line 3 would read

INIT 12345,67890

If the user wished he could have included (EXAMPLE) in every line.

Expansion of Macro

```
%EXPRO EX5
       PRINT 101
101    FORMAT('ABCDEF  ')
       PRINT 101
101    FORMAT('12345678')
```

Example 6. Use of Conditional Note and Comment Statement

This example demonstrates several features of the macro language. The first two model statements use the S-attribute and character-if statements to check to see if both arguments are present. The fourth line is standard, and the fifth terminates macro expansion. Line 6 is an internal comment and is not processed. Lines 7 and 9 generate macro error comments.

Generation of Procedure

```
(PRO)(EX6)
1(EXAMPLE)1 SQUARE X,Y
2(EXAMPLE) SQUARE M
3(EXAMPLE) SQUARE ,N
```

Listing of Procedure

```
%LIST EX6
       1      (EXAMPLE) SQUARE X,Y
       2      (EXAMPLE) SQUARE M
       3      (EXAMPLE) SQUARE
```

Generation of Macro

```
(MAC)(SQUARE)(EXAMPLE)
 MACRO WAS PREVIOUSLY DEFINED
1 &LABEL SQUARE &A,&B
2(MIFC) '@S&A' EQ '0' 7
3(MIFC) '@S&B' EQ '0' 9
4(FORTRAN)&LABEL &A=&B*&B
5(MEXIT)
6(MCOM) THE NEXT STATEMENT IS AN ERROR MESSAGE
7(MNOTE) "&&A" IS MISSING
8(MEXIT)
9(MNOTE) "&&B" IS MISSING
10(MEND)
```

Listing of Macro

```
%LIST SQUARE EXAMPLE
       1       &LABEL SQUARE &A,&B
       2      (MIFC) '@S&A' EQ '0' 7
       3      (MIFC) '@S&B' EQ '0' 9
       4      (FORTRAN)&LABEL &A=&B*&B
       5      (MEXIT)
       6      (MCOM) THE NEXT STATEMENT IS AN ERROR MESSAGE
       7      (MNOTE) "&&A" IS MISSING
       8      (MEXIT)
       9      (MNOTE) "&&B" IS MISSING
      10      (MEND)
```

Expansion of Macro

```
%EXPRO EX6
 1     X=Y*Y
 ***ERROR***
  "&B" IS MISSING
 ***ERROR***
  "&A" IS MISSING
```

Example 7. Use of Interpreter and Macro Processor Together

This last example uses the interpreter and macro processor together. Line 3 sets the interpreter in arithmetic mode. Line 2 checks for a missing argument, and line 4 uses the interpreter to find the list in the data structure.

In line 5, the return code from LIST is checked, and if it is non-zero (implying that the list cannot be found) an error message is printed. Line 6 calls FN to dump the list.

Generation of Macro

```
(MAC)(DUMP)(EXAMPLE)
1 DUMP &LIST
2(MIFC) '@S&LIST' EQ '0' 8
3(INTERP)(ARITHMETIC)
4(INTERP) LIST '&LIST' POINTER
5(MIFA) &SYSRTNCD NE 0 10
6(INTERP) FN POINTER 6F000000@@X 6F000000@@X
7(MEXIT)
8(MNOTE) NO ARGUMENT GIVEN
9(MEXIT)
10(MNOTE) LIST "&LIST." DOES NOT EXIST
11(MEND)
```

Listing of Macro

```
&LIST DUMP EXAMPLE
       1       DUMP &LIST
       2      (MIFC) '@S&LIST' EQ '0' 8
       3      (INTERP)(ARITHMETIC)
       4      (INTERP) LIST '&LIST' POINTER
       5      (MIFA) &SYSRTNCD NE 0 10
       6      (INTERP) FN POINTER 6F000000@@X 6F000000@@X
       7      (MEXIT)
       8      (MNOTE) NO ARGUMENT GIVEN
       9      (MEXIT)
      10      (MNOTE) LIST "&LIST." DOES NOT EXIST
      11      (MEND)
```

Generation of Procedure

```
(PRO)(EX7)
1(EXAMPLE) DUMP MLANGDIR
2(EXAMPLE) DUMP AAAAAAAA
3(EXAMPLE) DUMP
```

Listing of Procedure

```
%LIST EX7
       1      (EXAMPLE) DUMP MLANGDIR
       2      (EXAMPLE) DUMP AAAAAAAA
       3      (EXAMPLE) DUMP
```

Expansion of Macro

```
%EXPRO EX7
       DUMP OF MLANGDIR
       COMMAND   00519E78
       EXAMPLE   005010C8
***ERROR***
 LIST "AAAAAAAA" DOES NOT EXIST
***ERROR***
 NO ARGUMENT GIVEN
```

APPENDIX A.

Descriptions of subroutines used in the macro processor and available for general use.

NAME: CLNUMB

PURPOSE: to convert an internal line number to EBCDIC characters for printing.

CALLING SEQUENCE: CALL CLNUMB(VALUE,PTR,&1)

ARGUMENTS:

VALUE fullword integer internal line number

PTR fullword pointer to 12-byte output area

RETURN CODE:

RC=4 absolute value of VALUE too big for line number

COMMENTS:

1. the absolute value of VALUE must be less than 99,999,999.

2. the format of the output is ø-99999.999ø, where the 9's can be any numeric digit.

NAME:	CONIC4	
PURPOSE:	to convert a fullword integer to EBCDIC characters for printing	
CALLING SEQUENCE:	CALL CONIC4(PTR,LEN,VALUE,&1)	
ARGUMENTS:	PTR	fullword pointer to start of output region
	LEN	halfword integer maximum length of area
	VALUE	fullword integer value to be converted
RETURN CODE:	RC=4	insufficient room for sign and all digits
COMMENTS:	none	

NAME:	CONBH
PURPOSE:	to convert a binary string to hexadecimal characters for printing
CALLING SEQUENCE:	CALL CONBH(PTRIN,LEN,PTRO,&1)
ARGUMENTS:	PTRIN fullword pointer to string to be converted
	LEN halfword length of string
	PTRO fullword pointer to output region
RETURN CODE:	RC=4 length not positive
COMMENTS:	length of output region is twice the value in LEN

NAME: CONCN

PURPOSE: to determine a numeric constant type and convert accordingly.

CALLING SEQUENCE: CALL CONCN(PTR,LEN,IRES,RES,SW,&1)

ARGUMENTS:

PTR fullword pointer to start of string

LEN halfword maximum length of string

IRES result if type is integer (returned)

RES result if type is real (returned)

SW fullword switch indicating type (returned)

0 ==> fixed point
1 ==> floating point

RETURN CODE: RC=4 INVALID ARGUMENTS

COMMENTS: This subroutine determines if a string fits the criteria for an integer or real constant. It then calls either CONCF4 or CONIC4 to convert the string. The proper return argument is stored and the switch set to indicate the type.

NAME: CONCF4

PURPOSE: to convert a character string to a single-precision floating-point number

CALLING SEQUENCE: CALL CONCF4(PTR,LEN,RES,&1)

ARGUMENTS:

PTR — fullword pointer to start of string

LEN — halfword integer maximum length of string

RES — single-precision floating-point value of string (returned)

RETURN CODE:

RC=4 — string does not conform to FORTRAN IV standards for a double-precision floating-point number

COMMENTS:

1. Strings accepted are exactly those defined as REAL*8 constants in FORTRAN, except an 'E' is used for exponent notation rather than a 'D'. The resulting value is truncated to single-precision.

2. The string is scanned and as much of it as possible is used in the constant conversion. PTR and LEN are updated to the remainder of the string.

NAME: CLNUM

PURPOSE: to convert characters in MTS line number format to internal fixed point.

CALLING SEQUENCE: CALL CLNUM(PTR,IRES,PTRN,&1)

ARGUMENTS:

PTR fullword pointer to start of string

IRES fullword integer; on return it contains value of number X1000

PTRN fullword pointer to character on which scanning stopped (returned)

RETURN CODE:

RC=4 string does not fit definition of MTS line number

COMMENTS: The string is scanned to a break character and PTRN is set to point to that character.

NAME: CONFC8

PURPOSE: to convert a double-precision floating-point number to EBCDIC characters for printing.

CALLING SEQUENCE: CALL CONFC8(PTR,LEN,DEC,VALUE,&1)

ARGUMENTS:

PTR fullword pointer to start of output area

LEN halfword integer length of output area

DEC halfword integer maximum number of digits after decimal point.

VALUE REAL*8 value of number to be converted

RETURN CODE: RC=4 improper lengths (see comments)

COMMENTS:

1. The maximum number of digits before the decimal point is LEN-DEC-6.

2. DEC can be any integer from 0 to 12.

3. LEN can be any integer from DEC+6 to DEC+17; i.e., the number of digits before the decimal point can range from 0-11.

NAME: CONFC4

PURPOSE: to convert a single-precision floating-point number to EBCDIC characters for printing.

CALLING SEQUENCE: CALL CONFC4(PTR,LEN,DEC,VALUE,&1)

ARGUMENTS:

PTR	fullword pointer to start of output region
LEN	halfword integer length of area
DEC	halfword integer maximum number of digits after decimal point
VALUE	REAL*4 value of number to be converted

RETURN CODE:

RC=4	improper lengths(see comments)

COMMENTS:

1. The maximum number of digits before the decimal point is LEN-DEC-6.

2. DEC can be any integer from 0-12.

3. LEN can be any integer from DEC+6 to DEC+17; i.e., the number of digits before the decimal point can range from 0-11.

NAME:	CONIC2	
PURPOSE:	to convert a halfword integer to EBCDIC characters for printing	
CALLING SEQUENCE:	CALL CONIC2(PTR,LEN,VALUE,&1)	
ARGUMENTS:	PTR	fullword pointer to start of output region
	LEN	halfword integer length of output area
	VALUE	halfword integer value to be converted.
RETURN CODE:	RC=4	insufficient room for sign and all digits
COMMENTS:	none	

NAME: CONCI2

PURPOSE: to convert a character string to a halfword integer

CALLING SEQUENCE: CALL CONCI2(PTR,LEN,IRES,&1)

ARGUMENTS:

PTR fullword pointer to start of string

LEN halfword maximum length of string

IRES halfword integer result (returned)

RETURN CODE:

RC=4 string does not meet FORTRAN IV standards

COMMENTS:

1. Strings accepted are exactly those defined as INTEGER*2 constants in FORTRAN.

2. The string is scanned and as much of it as possible is used in the constant conversion. PTR and LEN are updated to the remainder of the string.

NAME: CONCI4

PURPOSE: to convert a character string to a fullword integer

CALLING SEQUENCE: CALL CONCI4(PTR,LEN,IRES,&1)

ARGUMENTS:

PTR fullword pointer to start of string.

LEN halfword maximum length of string

IRES fullword integer result (returned)

RETURN CODE:

RC=4 String does not meet FORTRAN IV standard for an integer constant.

COMMENTS:

1. Strings accepted are exactly those defined as INTEGER*4 constants in FORTRAN.

2. The string is scanned and as much as possible is used in the constant conversion. PTR and LEN are updated to the remainder of the string.

NAME: CONHB

PURPOSE: to convert a string of hexadecimal characters to binary equivalent

CALLING SEQUENCE: CALL CONHB(PTR,LEN,RESPTR,RESLEN,&1)

ARGUMENTS:

PTR fullword pointer to start of string

LEN halfword maximum length of string

RESPTR fullword pointer to location of output area

RESLEN halfword length of output area

RETURN CODE:

RC=4 LEN not > 0 and < 513; or RESLEN not > 0 and < 257

COMMENTS:

1. The converted binary string is right-justified and either padded with zeros or truncated to fit output area.

2. An odd number of hexadecimal digits is accepted.

3. As much of the string as possible is converted. PTR and LEN are updated to the remainder of the string.

NAME: CONCF8

PURPOSE: to convert a character string to a double-precision floating-point number

CALLING SEQUENCE: CALL CONCF8(PTR,LEN,RES,&1)

ARGUMENTS:

- PTR — fullword pointer to start of string
- LEN — halfword maximum length of string
- RES — double-precision floating-point value of string (returned)

RETURN CODE:

- RC=4 — string does not conform to FORTRAN IV standards for a double-precision floating-point number.

COMMENTS:

1. Strings accepted are exactly those defined as REAL*8 constants in FORTRAN, except an 'E' is used for exponent notation rather than a 'D'.
2. The string is scanned and as much of it as possible used in the constant conversion. PTR and LEN are updated to the remainder of the string

REFERENCES

1. Julyk, L.J., and Wolf, L.W., The CAMA Data Structure, Memorandum 29, Concomp Project, University of Michigan, Ann Arbor, August 1970.

2. Julyk, L.J., The CAMA Operating System, Memorandum 30, 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 Interpreter, ibid.

5. IBM Systems Reference Library C28-6514.

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 MACRO PROCESSOR

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Memorandum 35

5. AUTHOR(S) *(First name, middle initial, last name)*
T.J. Dingwall, L.J. Julyk, and L.W. Wolf

6. REPORT DATE
August 1970

7a. TOTAL NO. OF PAGES
31

7b. NO. OF REFS
5

8a. CONTRACT OR GRANT NO.
DA-49-083 OSA-3050

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
Memorandum 35

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

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

Under the CAMA (Computer-Aided Mathematical Analysis) System, we have devised a special macro processor which has as its object-languages not assembler languages, but higher-order languages such as FORTRAN, MAD, or ALGOL. The macro processor was designed to accomplish a number of objectives. First, it would enable a relatively unsophisticated user who is acquainted only with a language such as FORTRAN and not with the assembly language, to create macros and bases of languages without having to code in the assembly language. Second, the macro language was created to be a preprocessor for the interpreter language in CAMA. Third, it was created to be an intermediate processor between the mathematical expressions generated in the terminal computer which pass through a parsing operation and the base language such as FORTRAN. The macro processor was also created so that commands for the CAMA system could be written and extended easily, thereby enabling relatively unsophisticated users to extend the commands for their own particular needs, as well as write the original system commands.

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 macro processor macro processor						