

AD 733772



TM-4790/000/00

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
Springfield, Va. 22151

THE ASSISTANT MATHEMATICIAN

23 September 1971

DDC
RECEIVED
DEC 8 1971
B

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) System Development Corporation Santa Monica, California		2a. REPORT SECURITY CLASSIFICATION Unclassified	
		2b. GROUP	
3. REPORT TITLE The Assistant Mathematician (TAM)			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Technical Report - 23 September 1971			
5. AUTHOR(S) (First name, middle initial, last name) Joan Bebb			
6. REPORT DATE 23 September 1971	7a. TOTAL NO OF PAGES 33	7b. NO OF REFS 0	
8a. CONTRACT OR GRANT NO DAHC15-67-C-0149	8b. ORIGINATOR'S REPORT NUMBER(S) TM-4790/000/00		
9. PROJECT NO ARPA Order #1327, Amendment #3, Program Code #1D30, and IP10	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) None		
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY	
13. ABSTRACT TAM is an interactive program which performs arithmetic computations on constants, variables, and one- or two-dimensional arrays. TAM operates from two-dimensional, hand-written input and generates two-dimensional output. TAM is fundamentally a "one statement at a time" system. Each statement is accepted and, if legal, executed before the next statement is requested. However, statements can modify the TAM environment so that subsequent statements are affected. The program can recall and re-execute previous statements and has a powerful editing capability. Wherever possible, TAM frees the user from explicit or ordered declaration. Storage is acquired through usage; unknown quantities are requested when needed. TAM incorporates a powerful set of arithmetic operators on constants, variables, and one- and two-dimensional arrays. It also provides looping facilities, single statement functions, and user-defined input and output. Some built-in functions, such as logarithm, and some built-in constants, such as π and e , are provided.			

TECHNICAL MEMORANDUM

(TM Series)

The work reported herein was supported by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC15-67-C-0149, ARPA Order No. 1327, Amendment No. 3, Program Code No. 1D30, and 1P10, also supported by NASA Contract NAS12-526.

THE ASSISTANT MATHEMATICIAN

by

Joan Bebb

23 September 1971

SYSTEM
DEVELOPMENT
CORPORATION
2500 COLORADO AVE.
SANTA MONICA
CALIFORNIA
90406

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.



Distribution of this document is unlimited.

TABLE OF CONTENTS

<u>Section</u>		<u>Page</u>
1.	INTRODUCTION	1
2.	TAM ENTITIES	1
2.1	Quantities	1
2.2	Identifiers	2
3.	TAM OPERATORS	2
3.1	Operators	2
3.2	Operational Hierarchy	3
3.3	Operator Definition	3
4.	TAM STATEMENTS	4
4.1	Assignment Statement	5
4.1.1	Form	5
4.1.2	Use	5
4.2	Function Definition Statement	6
4.2.1	Form	6
4.2.2	Use	7
4.3	Input Statement	7
4.3.1	Form	7
4.3.2	Use	8
4.4	Output Statement	8
4.4.1	Form	8
4.4.2	Use	8
4.5	Loop Control	9
4.5.1	Forms	9
4.5.1.1	Form 1	9
4.5.1.2	Form 2	9
4.5.1.3	Form 3	9
4.5.2	Use	10
4.6	Built-in Functions	10

TABLE OF CONTENTS (cont'd)

<u>Section</u>		<u>Page</u>
5.	INPUTTING AND EDITING TAM STATEMENTS	11
5.1	Overview	11
5.2	Pushbuttons	12
6.	UNCONSTRAINED TAM	14
6.1	Minimum Declaration	14
6.2	Demand Input	15
6.3	Natural Expression of Operation	15
6.3.1	Implicit Multiplication	16
6.3.2	Imbedded Exponentiation	16
7.	ERROR CONDITIONS	17
7.1	User Errors	17
7.2	Interpreter Errors	19
7.3	Capacity-Overload Errors	20
APPENDIX A	BUILDING A TAM DICTIONARY	22-26
APPENDIX B	CHARACTER CODES - STRING-PARSER-GENERATED CHARACTERS	27
APPENDIX C	CHARACTER CODES - LEGAL USER CHARACTERS	28-30

1. INTRODUCTION

TAM is an interactive program which performs arithmetic computations on constants, stored variables, and one- or two-dimensional arrays. TAM operates from two-dimensional, hand-written input and generates two-dimensional output.

TAM is fundamentally a "one statement at a time" system. Each statement is accepted and, if legal, executed before the next statement is requested. However, statements can modify the TAM environment so that subsequent statements are affected. The program can recall and re-execute previous statements and has a powerful editing capability.

Wherever possible, TAM frees the user from explicit or ordered declaration. Storage is acquired through usage; unknown quantities are requested when needed.

TAM incorporates a powerful set of arithmetic operators on constants, variables, and one- and two-dimensional arrays. It also provides looping facilities, single statement functions, and user-defined input and output.

Some built-in functions, such as logarithm, and some built-in constants, such as π and e , are provided.

2. TAM ENTITIES

2.1 QUANTITIES

Quantities in TAM are either positive or negative, integral or mixed numbers. Internally, mixed numbers are carried in double-precision, floating-point form; integers are carried in one IBM/360 word (4 bytes). Therefore, effective precision is that defined for the IBM/360. Quantities may be contained in variables or arrays or expressed as constants. Most storage declaration is implied by usage. Arrays are dimensioned either implicitly or explicitly.

Examples: 3 5.7 $\begin{bmatrix} 1.0 & 2 \\ 3 & .45 \end{bmatrix}$ \emptyset -.37

2.2 IDENTIFIERS

Variables and array identifiers are single-letter names. The legal alphabet of TAM consists of Greek and Roman uppercase and lowercase letters. An identifier may be made unique through the use of overscoring or underscoring. Legal overscore and underscore characters are:

Examples of legal identifiers are:

a	<u>A</u>	α
A	<u>A</u>	$\dot{\alpha}$

3. TAM OPERATORS

3.1 OPERATORS

Quantities may be manipulated through the use of operators as follows:

$a + b$	addition
$a - b$	subtraction
$ab, a.b, a*b$	multiplication
$\frac{a}{b}, a/b$	division
X^Y	exponentiation
$\sqrt[n]{\quad}$	n^{th} root $\sqrt{\quad} \Rightarrow \sqrt[2]{\quad}$
!	factorial
{ }	absolute value

{ }	ceiling
{ }	floor
$\prod_{i=m}^n$	product
$\sum_{i=m}^n$	summation
+b, -b	unary sign
T	transpose (two-dimensional arrays only)

Note that implicit multiplication is allowed because all identifiers are single letters (possibly qualified). A special operator, \mathbf{V} , is used in conjunction with setting arrays and is explained subsequently.

3.2 OPERATIONAL HIERARCHY

Parenthesization is allowed to control the parsing hierarchy. In the absence of parenthesization, the hierarchy of operation is (from least to most binding):

\sum	summation
\prod	product
+, -	addition, subtraction
*, /	multiplication, division
+, -	unary plus, unary minus
!	factorial
{ }, { }, Ln, [], $\sqrt{\quad}$, $\sqrt[n]{\quad}$, T	floor, ceiling, function call, absolute value, root, exponentiation, transpose

3.3 OPERATOR DEFINITION

Each operator is usable when meaningful. With few exceptions (for example, transpose applies to matrices only; $(-3)!$ is signaled as an error), all operators are usable to manipulate single constants or variables. The operators

are legal when applied to arrays where an acceptable matrix or vector operation is defined. For example,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-3} = \left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^3 \right)^{-1} \quad \text{is defined if the matrix is square;}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{1/3} \quad \text{is not defined;}$$

$$\frac{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}{\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}^{-1} \quad \text{is defined if the divisor is a square matrix and the interior dimensions of the two matrices are the same;}$$

and

$$\begin{bmatrix} 1 & 5 \\ 7 & 9 \end{bmatrix} ! \quad \text{is not defined.}$$

One-dimensional arrays are stored and treated as row vectors, with one exception. In multiplication, if one or both operands are vectors, the operand on the left (if a one-dimensional array) is treated as a row vector and the operand on the right (if a one-dimensional array) is treated as a column vector. The multiplication performed is the dot product. Therefore,

$$[1 \ 2 \ 3] * \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 14$$

4. TAM STATEMENTS

There are five distinct TAM statements: assignment, function definition, input, output, and loop.

23 September 1971

4.1 ASSIGNMENT STATEMENT

The assignment statement is used to set identifiable variables or arrays, presumably for use in subsequent statements.

4.1.1 Form

identifier + expression

The expression may consist of any legal manipulation of quantities.

4.1.2 Use

For example:

$\hat{A} \leftarrow 3$ Set variable \hat{A} to 3. (1)

$B \leftarrow \begin{bmatrix} 1 & 2 & \hat{A} \\ 4 & 5 & 6 \end{bmatrix}$ Set array B to the given matrix (also (2)
dimension B as 2, 3). Each element
of the matrix B may be any legal
expression that yields a single numeric
value.

$C \leftarrow V_{3,5}^{\emptyset}$ Set all elements of array C to \emptyset (3)
dimension C 3, 5.

$D_1 \leftarrow 25.5$ Set first element of array D to (4)
25.5. The dimensionality of D is
unknown as yet.

$X \leftarrow -\sum_{j=1}^{\hat{A}} B_{1,j}$ Set X to minus the sum of the (5)
elements of the first row ($\hat{A}=3$) of B.

$$Y = D_1^{-X} + |X| \sqrt{B_{2,2}} \quad \text{Set } Y \text{ to the sum of } D_1 \text{ to the } -X^{\text{th}} \quad (6)$$

power and the product of the absolute
value of X and the square root of $B_{2,2}$.

The third example illustrates the use of the special operator \mathbf{V} . It is used to declare, dimension, and preset an array (of one or two dimensions). The form of the operator is:

$$\mathbf{V}_{s,t}^p$$

where s and t are dimensions (t and the preceding comma are optional) and p (also optional) is the presetting value. P may be any legal expression that yields a single numeric value.

4.2 FUNCTION DEFINITION STATEMENT

The TAM user may define frequently used arithmetic expressions as functions; he may then call upon these functions when necessary. Functions, of course, return values. The function definition and call may contain parameters. Both the function expression and the actual parameters of the call may contain calls to other functions.

4.2.1 Form

$f_n(p_1, p_2, \dots, p_m) = \text{expression}$
 where f is a legal identifier, n is an optional alphabetic or numeric qualifier, and the p_i are optional parameters. The expression may involve any legal manipulation of quantities. The identifier f , once it has been used as a function name, defines a class of functions f_n and cannot be later used as a variable or array identifier. The various functions in class f are distinguished from one another through the use of the qualifier n . For example $\bar{G}_1(X) = X^2$ and $\bar{G}_2(X) = X$ are two functions in class \bar{G} ; $\bar{G} + 3$ is an illegal statement. \hat{G} and G are not in class \bar{G} . As many function classes as desired

may be defined. The optional parameters, p_1 , must be legal identifiers. The same identifier may be used as a parameter in many function definitions and also as a variable or array name or as a function class.

4.2.2 Use

An example of a function definition and call:

Function definition:

$$E_3(a,b) = \frac{\{a\}}{\sqrt[3]{b}}$$

Function call:

$$\theta + E_3(27.2, \hat{A}) + Y$$

To exponentiate the value returned by a function, it is permissible to write (for function a, parameters b,c, exponent 2):

$$a^2(b,c)$$

or

$$a(b,c)^2$$

4.3 INPUT STATEMENT

Because TAM requests values for undefined quantities as they are encountered, user-directed input is seldom necessary. However, an input statement is provided so that the user may guide the order of quantity setting in a direct fashion and so that he may reset quantities easily.

4.3.1 Form

$\square \rightarrow$ list

where the list consists of legal identifiers of simple variables or arrays separated by commas.

4.3.2 Use

The statement results in a request for input(s) from the user. A value for each element in the list is requested in turn. For example:

$$\square \rightarrow \Omega, \alpha$$

results in the user being prompted with:

$$\Omega \leftarrow$$

Hopefully, the user will then respond with a value for Ω .

TAM will then request a value for α by writing:

$$\alpha \leftarrow$$

Once again, the user is expected to input a value for α .

4.4 OUTPUT STATEMENT

4.4.1 Form

list

where the list consists of expressions or identifiers separated by commas.

4.4.2 Use

The statement:

$$B, 3 + 5$$

results in the output

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

8

assuming B has been defined as the given matrix. If B has not yet been defined, the following, rather unusual, result occurs. The user is first asked to input a value for B by the prompt:

$$B \leftarrow$$

After receiving B, TAM outputs it.

4.5 LOOP CONTROL

An assignment statement, input statement, or output statement may be iterated by following the statement with loop-control information.

4.5.1 Forms

Loop control may be specified in three forms. Loops may be nested to any level, but each loop variable in the nest must be unique.

4.5.1.1 Loop Control, Form 1

Statement: $i=m, \dots, n$

where i is the loop variable (an identifier of a simple variable whose value will be incremented by one for each iteration of the statement), m is the initial value for i , and n is the terminal value. m and n may be any legal expressions that yield single numeric values. The iteration is complete when i exceeds n . The statement iterated may, but need not, contain references to i .

4.5.1.2 Loop control, Form 2

Statement: $i=m_1, m_2, \dots, n$

where i is the loop variable, m_1 and m_2 are the first two values for i as the statement is iterated, $m_2 - m_1$ defines the loop increment (or decrement), and n is the terminal value. m_1, m_2 and n may be any legal expressions which yield single numeric values. The iteration is complete when i exceeds (or becomes less than) n . The statement iterated may, but need not, contain references to i .

4.5.1.3 Loop Control, Form 3

Statement: $i=m_1, m_2, m_3, m_4, \dots, m_n$

where i is the loop variable and the m_j are successive settings for i each time the statement is iterated. The ellipsis (...) shown is not a part of the

loop-control form, as it is in the two previous forms, but is included to indicate that the list m_j is of user-determined length. The loop terminates after the statement has been executed for $i=m_n$. The statement may, but need not, contain references to i .

4.5.2 Use

For example:

$$X_{1,j} + Y + Q: i=1, \dots, 5; j=2, 4, \dots, 8$$

is an instance of a two-level nest of loop control. The level of nesting proceeds from right to left, i.e., j is the outer control variable, i is the inner control variable.

Another example:

$$A \rightarrow A + B: B = 86\sqrt{m}, 4.7, i + j$$

B assumes the values $86\sqrt{m}, 4.7, i + j$ for successive iterations of the loop. The final value for A is the sum of these values plus the original contents of A .

4.6 BUILT-IN FUNCTIONS

TAM includes a set of built-in functions that the user can activate by including one of the names given below (along with an appropriate parameter) within any context in which a function call is permissible. (In expressing the name, any combination of uppercase and lowercase Roman letters is permissible; e.g., $Ln=ln=LN$.) The available functions are:

<u>Name and Parameter</u>	<u>Definition</u>
$\sin(x)$	sine(x)
$\cos(x)$	cosine(x)
$\tan(x)$	tangent(x)
$\cot(x)$	cotangent(x)
$\arctan(x)$	arctangent(x)
$\tan^{-1}(x)$	arctangent(x)
$\ln(x)$	natural logarithm(x)

The arguments are in radians for the trigonometric functions.

5. INPUTTING AND EDITING TAM STATEMENTS

5.1 OVERVIEW

TAM statements are input in hand-written two-dimensional form through a RAND-Tablet display device. TAM uses a previously constructed dictionary of patterns describing the user's individual handwriting characteristics and defining the recognizable character set (see Appendix A).

Once active at the terminal, TAM asks for a file description of this dictionary:

OLD DICTIONARY =

The user then responds with the appropriate file description.

TAM then displays a set of "pushbuttons" (e.g., the words "Exit" and "Erase") that allow the user to direct subsequent operations. The TAM pushbuttons and their meanings are given in Section 5.2.

To start operating, the user writes a TAM statement on the face of the display. The line will be parsed as recognized and an encoded form of the line will appear (see Appendix B for encoding characters). The user can then operate on his statement through use of the pushbuttons. As TAM operates, a history display is built of the last four legal lines and any associated output. Any of the last four lines entered can be recalled for editing and/or reoperation.

5.2 PUSHBUTTONS

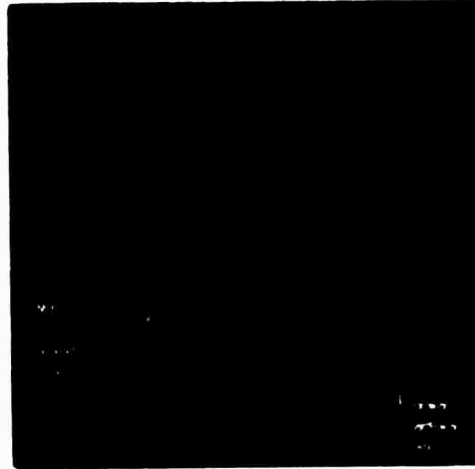


Figure 1. TAM Pushbuttons

<u>Button</u>	<u>Function</u>
Erase	The most recently input user characters, their related parsed display, and any associated record in the interpreter interface are deleted. A new or recalled line is now expected.
Move	The designated user characters are moved as specified by the editing stroke(s). The move operations and how they are indicated are: <u>Open-up</u> --a single horizontal line drawn on or between previously scripted characters. A space equal to the length of the line is made between the selected characters. The initial point of the line determines the location of the space and the direction of the line determines the direction in which the affected characters are moved. <u>Close-up</u> --two horizontal lines, similar to an equal sign, drawn in a space between previously scripted characters. The

<u>Button</u>	<u>Function</u>
Move (cont'd)	characters to the right of the editing strokes are moved to the left a distance equal to the length of the longest of the editing stroke pair.
	<u>Move character</u> --a circle or a rectangle, drawn as one stroke, followed by a single line. The characters enclosed by the circle or rectangle are moved the length and direction of the line.
Context	This button may be used in conjunction with the Move-character operation and the character scrub (the latter consists of rubbing out characters with the pen). If the Context button is pushed immediately prior to the scrub action, any characters logically attached to the scrubbed character, such as a superscript, are erased. If the Context button is used in place of the Move button, any characters logically attached to a character enclosed by the first stroke of the move character are also moved.
TAM	The current user characters are processed by the interpreter. If no characters exist, the action is taken as a cancel signal. When the current line becomes the history display, the program is ready for new input. The interpreter's output, if not an error message, is joined with the parsed representation of the user's scripted input and added to the history display, and the previous entry in the history display is deleted. An internal program history pointer is set to the most recent history entry.
Latest	The user characters in the history display are regenerated and displayed.

<u>Button</u>	<u>Function</u>
Backward	The history pointer is moved from its current position to the next older entry. The entry's user characters are then regenerated and displayed.
Forward	The history pointer is moved from its current position to the next more recent entry. The entry's user characters are then regenerated and displayed.
Quit	Terminate TAM.

6. UNCONSTRAINED TAM

6.1 MINIMUM DECLARATION

One of the important design goals of TAM is that the user be freed from unnecessary or ordered declaration of storage, quantities, or functions. One manifestation of this freedom is that an array of elements need not be explicitly dimensioned until the user wishes to operate with the array as an entity, i.e., as a matrix or vector; so long as the user deals with his array element by element, no explicit dimensioning is necessary. Before the array is first used as a whole, the user must dimension it. Then, when the array is used, the user is asked to set elements that do not yet have values. In setting array values, the user may either override or maintain existing values.

TAM gives the user this freedom by incorporating the following techniques. An identifier--a single-letter name along with any overscore and/or underscore characters expressed with the letter--is classified into one of two classes, function or variable, on the basis of its first usage. That is, the four distinct identifiers

A \bar{A} \underline{A} $\underline{\bar{A}}$

will be separately classified according to how

23 September 1971

15

they are first used. Subscripts used with identifiers discriminate between class members by type. Each member of a class has associated with it a type: list, vector, array, or function.

For example:

$\dot{A}_1 \leftarrow 3$ sets \dot{A} to class 'variable' and \dot{A}_1 to type 'list'.

Before \dot{A} can be used as a vector, dimensioning information must be given about \dot{A} . The type of \dot{A} there will be set to 'vector'. If necessary, TAM will demand values for unset members of class \dot{A} when it is used.

6.2 DEMAND INPUT

TAM further frees the user from ordered statements in that quantities to be manipulated need have values only when used. TAM automatically provides the necessary prompting. For example:

$A \leftarrow \hat{Z} + Y$

as the first TAM statement would result in the request

$\hat{Z} \leftarrow$

to which the user would be expected to respond with a value or expression. If the user's response contains unknown quantities, TAM also requests values for these. Once a value for \hat{Z} is ascertained, TAM requests Y :

$Y \leftarrow$

Once again, a value or expression is mandatory. Once all quantities are defined, A is set to the value of the original expression $\hat{Z} + Y$.

6.3 NATURAL EXPRESSION OF OPERATION

TAM allows the user to express standard mathematical operations in a natural way. Two-dimensional input is TAM's outstanding facility in this regard. In

addition, two features of standard notation usually disallowed in programming languages are permitted in TAM: implicit multiplication and embedded exponentiation in function calls.

6.3.1 Implicit Multiplication

TAM allows the user to imply the multiplication operator. The nature of TAM identifiers eliminates contextual confusion. The following are examples of implied multiplication:

$$ab \Rightarrow a*b$$

$$c(de) \Rightarrow c*(d*e)$$

$$e \sum_{i=3}^5 14 \Rightarrow e * \sum_{i=3}^5 1*4$$

Note: $ij! \Rightarrow i*(j!)$

$$(ij)! \Rightarrow (i*j)!$$

and $ab^2 \Rightarrow a*(b^2)$

$$(ab)^2 \Rightarrow (a*b)^2$$

6.3.2 Embedded Exponentiation

Exponentiation of the value returned by a function can be expressed in either of two ways:

$$\sin (a)^2 \Rightarrow (\sin(a))^2$$

or

$$\sin^2(a) \Rightarrow (\sin(a))^2$$

That is, the exponent may appear immediately after the full function call (which includes parameters) or immediately adjacent to the function name, before the parameters.

7. ERROR CONDITIONS

TAM legality checks each statement before attempting to execute it. If an illegality is discovered, the statement is rejected and an appropriate message is generated.

7.1 USER ERRORS

The most frequent error message generated by TAM is:

SYNTAX ERROR AT CHARACTER XX

where XX is the relative character position (in the parsed string) of the illegality. Other possible error messages are:

MessageMeaning

TYPE ERR

1. An identifier defined as a function name is being used as a variable.
2. An identifier defined as a variable or array name is being used as a function.
3. The control variable in a loop statement is an array or function name.

X! $X \neq \text{INTEGER} \geq 0$

The factorial operator is defined for positive integers only.

CLASS ERR

1. An identifier which is a member of a class of functions is being used as a variable.
2. An identifier which is in a variable or array class is being defined as a function.

Message

OPERATION ERR - ARRAY/VECTOR

X^Y , $X < 0$ $Y \neq \text{INTEGER}$

PARAMETER ERR

\neq COLUMNS/ROWS

SUBSCRIPTS MUST BE INTEGERS

SINGULAR MATRIX, NO SOLUTION

LOG OF - OR 0 ARG RETURNS - INFINITY

SYMBOL OR STRING TOO LONG

Meaning

An operator is being used which is meaningless if one or both operands is an array or vector.

1. X^Y , Y is a vector or array.
2. X^Y , Y is not an integer, X is an array or vector,
3. An array or vector name is being used as a subscript.

For negative X, only exponentiation to an integral power is defined.

The number of parameters in a function call does not match the number of parameters in the definition.

In a block input of an array, the columns are of unequal length or the rows are of unequal width.

An attempt to invert a singular matrix has been made.

$\text{Ln}(X)$, $X \leq 0$
returns a very small number.



MessageMeaning

MISMATCHED DIMENSIONS

An arithmetic operation on two arrays has been attempted. The nature of the arrays precludes the operation.

EXPONENT OVERFLOW EXCEPTION

EXPONENT UNDERFLOW EXCEPTION

SIGNIFICANCE EXCEPTION

FLOATING DIVIDE EXCEPTION

FLOAT /= NO.

An array or vector has been specified in a context that demands a simple variable, e.g., as a preset value for an array or vector.

7.2 INTERPRETER ERRORS

The following messages should not, but may, appear. The program will stop. The messages signal errors in the TAM interpreter. If one of the following messages is encountered, a register and core dump should be taken and the TAM maintenance personnel contacted.

AN UNDEFINED ROUTINE HAS BEEN CALLED

BAD CAR

BAD CDR

GENERATOR DOES NOT MATCH ITS PARAMETERS

ALL BOOLEANS OF AN EXPRESSION ARE FALSE

BAD CADDR

BAD CADDR2

23 September 1971

20

System Development Corporation
TM-4790/000/00

BAD CADDR3
BAD CADDR4
BAD CADDR
BAD CADDR2
BAD CADDR3
BAD CADR
BAD CADR2
BAD CADDR
BAD CADDR2
BAD CADDR3
BAD CADDR4
BAD CADDR
BAD CADDR2
BAD CADDR3
BAD CDDR
BAD CDDR2
BAD CDDR3
BAD CDDR
BAD CDDR2
X NOT A CHARACTER
BAD COMPRESS OF ATOM
BAD OUTNUM2
ROOF x y COMPILER ERROR
BAD CONS
BAD DEF1
BAD DEF2
GETT x COMPILER ERROR
NOT ENOUGH SPACE
COMPILER ERROR HALT

7.3 CAPACITY OVERLOAD ERRORS

The following messages signal that the capacities of TAM have been exceeded. The program will stop. If one of these messages is encountered and the excess seems unwarranted or ill defined, a core and register dump should be taken and TAM maintenance personnel contacted.

23 September 1971

21

System Development Corporation
TM-4790/000/00

FULL UNDEFINED STACK
FULL BACKUP STACK
FULL ARGUMENT STACK
NO MORE ARRAY SPACE
NOT ENOUGH ARRAY SPACE
NOT ENOUGH LIST SPACE
NO MORE LARGE NUMBER SPACE
NO MORE REAL NUMBER SPACE
NO MORE GENSYM SPACE
NOT ENOUGH STRING SPACE

APPENDIX ABUILDING A TAM DICTIONARY

To operate TAM, the user must have, at some previous time, built a dictionary of recognizable characters. This dictionary provides the information to allow TAM to recognize and accept the user's individual handwriting and character set. Each character to be used with TAM must be entered into the dictionary.

A program, BUILDER, is provided for dictionary construction. The program will either create a new dictionary or allow the user to modify and amplify an existing one.

Using the Dictionary Builder

After signing on, the program asks:

CONTINUE W/ OLD DICT ? Y/N

A "Y" answer will cause the program to request file description information and open an old dictionary file. An "N" answer causes the program to assume a new dictionary is being built.

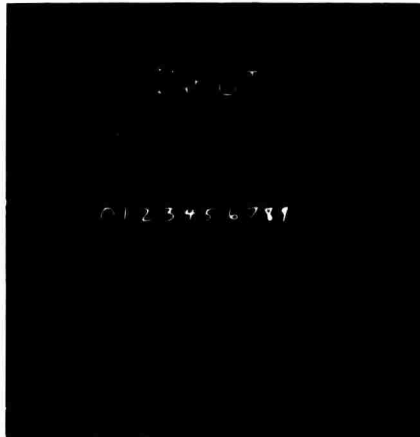
Function pushbuttons will then appear on the display. The pushbuttons and their functions are:

<u>Pushbutton</u>	<u>Function</u>
Save	Save the dictionary on disc or tape
Build	Build a dictionary
Test	Test the dictionary definitions
Summary	Print a summary of the dictionary contents
Print	Print a detailed description of the dictionary
Quit	Quit the program

(Only the Build and Save functions will be described.)

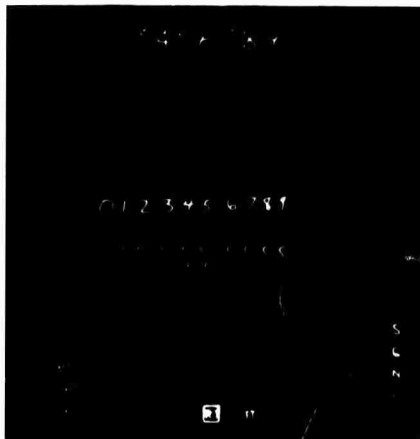
Build Function

The user builds a dictionary containing stroke information and the definition associated with the strokes. The stroke information consists of the user's own individual characteristics in drawing characters.



Sampling Mode:

The program asks for user input by displaying "INPUT" at the top of the display. The user then inputs the strokes he wants to define. Note that the strokes need not be similar to each other.



Keyboard

Input strokes
Dictionary matches/no matches

Program/
Control
Pushbuttons

Keyboard pushbuttons

Threshold pushbuttons

Keyboard, keyboard pushbuttons. There are five keyboards, each consisting of a unique set of definitions. The user selects the set containing the definitions he needs by touching one of the keyboard pushbuttons: "S" for special characters, such as the asterisk and plus sign, "G" for Greek characters, "N" for numerals 0 through 9, "U" for upper case Roman alphabet, and "L" for lower case Roman alphabet.

The uppercase Roman alphabet is the first keyboard set displayed; the last set selected will be displayed at the top of the display until another set is selected.

Dictionary matches/no matches. Each stroke is analyzed and compared to definitions already in the dictionary. If a match has been found (matches are subject to threshold values described below), the character associated with that match is displayed directly below the stroke. This character will either belong to one of the keyboard sets or be a backward "S". The backward "S" represents a match in the dictionary that has no character definition, i.e., is only a part of a defined character.

If no match has been found, a backward (to distinguish it from the character "?") question mark is displayed directly below the stroke.

These defines/undefines are displayed about an inch below the string of input strokes, each define/undefine aligned with its stroke, at successively lower levels to prevent overlapping of definitions.

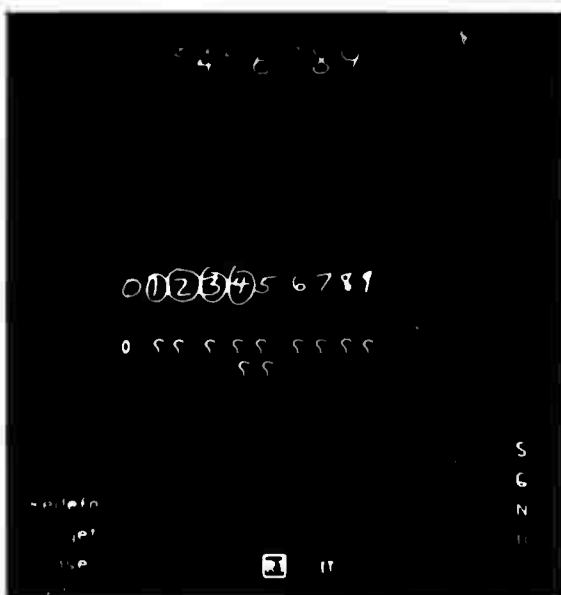
If the strokes are input below the bottom third of the screen, or if the layering of defines/undefines results in the overlapping of pushbuttons, then the defines/undefines appear above the threshold pushbuttons in the order in which their corresponding strokes were drawn.

Threshold pushbuttons. These buttons appear at the bottom center of the display, "RT" for reduced thresholds and "IT" for increased thresholds. A box appears around the threshold button last selected; the reduced thresholds are in effect first. These thresholds are used in determining whether or not a match exists for a given stroke/strokes.

Program/Control pushbuttons.

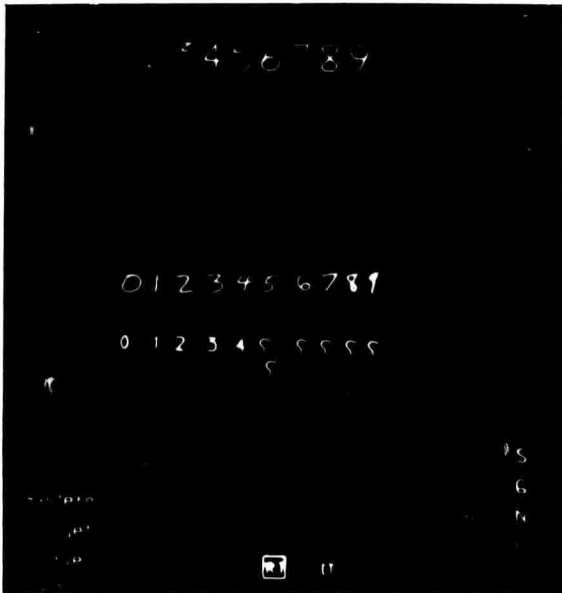
Redefn	Redefine button, allows the user to change a dictionary definition.
Forget	Forget button, ignore a previous action.
Erase	Clear the display and return to the sampling mode ("INPUT" displayed.)
Exit	Leave the Build function and return to function selecting mode.

Defining Mode.



A definition is entered in the dictionary by circling the stroke/strokes or either end point of a stroke and touching the corresponding character in the keyboard.

These definitions may be entered one at a time or as many as the user wishes, subject to program limitations.



The matches/no matches appear again below the input strokes. There may be all matches or there may still be some failures to match, as shown at the left.

The user may define the strokes for which no matches exist until all have matches in the dictionary.

Note: The possible matches are done using the thresholds in effect. Increasing the threshold values by touching the "IT" pushbutton may result in more matches.

To change the current definition in the dictionary for a certain stroke or strokes, the user should first touch the "Redefn" pushbutton and then define the stroke or strokes in the manner described above.

When satisfied that the current set of strokes has been properly defined, the user should touch either the "Erase" pushbutton to input a new set of strokes or the "Exit" pushbutton to leave the Build mode.







Save Function

To save the dictionary just built (for future use with TAM or for subsequent amplification), the user should touch the "Save" pushbutton. The program then requests a file identification (on the terminal). If a new file is to be opened, 'N' must be included in the file identification. Once the dictionary has been saved successfully, the message "DICT SAVED" appears on the terminal.

APPENDIX BCHARACTER CODES - STRING-PARSER-GENERATED CHARACTERS

<u>HEX</u>	<u>Octal</u>	<u>Decimal</u>	<u>Parse</u>	<u>Symbol</u>
C1	301	193	↑	Overscore; Upper Integral limit
C2	302	194	↗	Superscript
C3	303	195	↘	Subscript
C4	304	196	↓	Underscore, Lower Integral limit
C5	305	197	↙	Sq. Rt. index
D1	327	209	[Left Bracket
D2	322	210]	Right Bracket
D3	323	211	⌈	Numerator and Denominator enclosures
D4	324	212	⌋	
D5	325	213	⌈	Multiple overscore enclosures
D6	326	214	⌋	
D7	327	215	⌈	Matrix enclosures
D8	330	216	⌋	
D9	331	217	⌈	Combinatorial enclosures
DA	332	218	⌋	
EO	340	224	—	Square Root Bar
FO	360	240	,	Matrix Element Separator
F1	361	241	⁄	Fraction line
F2	362	242	⌈	Multiple overscore indicator
F3	363	243	×	Multiply cross
F4	364	244	⌋	Multiple underscore indicator
F6	366	246	;	Matrix Row Separator
FD	375	253	String	Start
FE	376	254	String	End

APPENDIX CUSER INPUT CHARACTERS

<u>Octal</u>	<u>Hex</u>	<u>Character</u>	<u>Octal</u>	<u>Hex</u>	<u>Character</u>
000	00	n/a	040	20	Blank
001	01	α	041	21	!
002	02	β	042	22	"
003	03	α	043	23	#
004	04	δ	044	24	\$
005	05	ε	045	25	%
006	06	φ	046	26	&
007	07	η	047	27	~
010	08	λ	050	28	(
011	09	μ	051	29)
012	0A	γ	052	2A	*
013	0B	π	053	2B	+
014	0C	ρ	054	2C	,
015	0D	σ	055	2D	-
016	0E	τ	056	2E	.
017	0F		057	2F	/
020	10		060	30	0
021	11		061	31	1
022	12		062	32	2
023	13		063	33	3
024	14	Σ	064	34	4
025	15	Γ	065	35	5
026	16	Δ	066	36	6
027	17	θ	067	37	7
030	18	Ω	070	38	8
031	19	χ	071	39	9
032	1A	π	072	3A	:
033	1B	{	073	3B	;
034	1C		074	3C	<
035	1D	}	075	3D	=
036	1E	!	076	3E	>
037	1F	~	077	3F	?

(Cont'd)

<u>Octal</u>	<u>Hex</u>	<u>Character</u>	<u>Octal</u>	<u>Hex</u>	<u>Character</u>
100	40	@	140	60	∞
101	41	A	141	61	a
102	42	B	142	62	b
103	43	C	143	63	c
104	44	D	144	64	d
105	45	E	145	65	e
106	46	F	146	66	f
107	47	G	147	67	g
110	48	H	150	68	h
111	49	I	151	69	i
112	4A	J	152	6A	j
113	4B	K	153	6B	k
114	4C	L	154	6C	l
115	4D	M	155	6D	m
116	4E	N	156	6E	n
117	4F	O	157	6F	o
120	50	P	160	70	p
121	51	Q	161	71	q
122	52	R	162	72	r
123	53	S	163	73	s
124	54	T	164	74	t
125	55	U	165	75	u
126	56	V	166	76	v
127	57	W	167	77	w
130	58	X	170	78	x
131	59	Y	171	79	y
132	5A	Z	172	7A	z
133	5B	[173	7B	↙
134	5C	\	174	7C	≤
135	5D]	175	7D	†
136	5E	↑	176	7E	≥
137	5F	+	177	7F	EOM

(Cont'd)

<u>Octal</u>	<u>Hex</u>	<u>Character</u>
200	80	reserved
201	81	[
202	82]
203	83	[
204	84]
205	85	{
206	86	}
207	87	~
210	88	^
211	89	□
212	8A	V