285
6-1-65
MASTER

Argonne National Laboratory

NAMED STORAGE

A Dynamic Storage-Allocation Scheme with Manipulative Routines

by

Stanley Cohen

PATENT CLEARANCE OBTAINED. RELEASE TO
THE PUBLIC IS APPROVED. PROCEDURES
ARE ON FILE IN THE RECEIVING SECTION.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

ANL-7021
Mathematics and
Computers
(TID-4500, 39th Ed.)
AEC Research and
Development Report

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60440

NAMED STORAGE
A Dynamic Storage-Allocation Scheme
with Manipulative Routines

by

Stanley Cohen

Physics Division

April 1965

Operated by The University of Chicago
under
Contract W-31-109-eng-38
with the
U. S. Atomic Energy Commission

TABLE OF CONTENTS

	Page
1. INTRODUCTION	2
2. THE ALLOCATOR	4
A description of the allocator and the functions of each of its entries.	
3. MANIPULATIVE ROUTINES	9
A general description of the operation of a manipulator.	
4. GENERAL-AREA MANIPULATORS	11
Specific manipulator functions available for handling general areas.	
5. ELEMENTAL MANIPULATORS	13
Manipulators for operations involving single elements.	
6. INPUT-OUTPUT MANIPULATORS	13
7. MATRIX AND VECTOR MANIPULATORS	16
A class of manipulators for vector and matrix operations.	
8. FUNCTION MANIPULATORS	20
A class of manipulators for performing operations on functions of one variable.	
9. PSEUDO ARRAYS	22
Description of a means of using conventional notation and at the same time integrating with named storage.	
10. REFERENCING AND ERROR DETECTION	23
The means by which manipulators interrogate the allocator for information—the automatic diagnostic facilities.	
11. NAMING LOCAL AREAS	26
12. EXAMPLE OF A MANIPULATOR	28
13. CONCLUDING REMARKS	29
APPENDIX: LIST OF STANDARD ENTRIES	32

1. INTRODUCTION

In writing very large programs for a modern computer, much effort is usually devoted to making efficient use of the storage facilities of the computer complex. Modern FORTRAN has available provisions such as variable dimensioned arrays and labeled COMMON. These are provided to allow the programmer flexibility in the use of the fast storage areas and the means for optimizing their use. However, neither is really sufficient to allow general optimization of the use of even the fast memory for very large problems. This report describes a dynamic storage allocator which is intended to provide more complete control. As a consequence of the form of this allocator, it is in fact possible to write programs that are not only more compact but that can be performed more efficiently than most programs written in the usual FORTRAN form. Basically, the approach taken here is intended to divorce those parts of a program concerned with data storage and manipulation from those involved in setting up the logic of the actual calculation. All tasks involving manipulation of data are assigned to standard functions of the system. The task of writing a computational program is then for the most part one of joining together the many subroutines of the system.

Efficient use of the storage facilities of a computer necessitates making provision for using the same parts of the memory for different purposes in various phases of calculation. Ideally the use of storage areas should be controlled by the operating program itself. Here we describe the use of a storage allocation scheme which allows the operating program to exercise complete control over the use of the storage facilities. It is possible to make use of the storage area in a way that not only adjusts to the size of the problem being treated, but that alters the allocation of space in the memory to suit the particular phases of the computation.

All of the features of the programming system to be described rely on the concept of NAMED STORAGE. Named storage means that all stored data can be referred to by name alone. All other information normally associated with particular data is made available as part of the functions of the system. With this extra information that is made available, standard system subroutines are able to perform tasks that normally are considered parts of a computational program. This greatly eases the task of writing the computational program since operations are performed with a minimum of transmitted information. Also, a great deal of extra checking for possible conflicting requests can automatically be carried out within the individual system subroutines since they have more information available than in the normal case.

As an example, the system subroutine for forming the inner product of two vectors need only be given the names of the two vectors. The routine itself can locate the two vectors, ascertain whether or not they are the same length, and determine whether they are floating-point or fixed-point vectors. It can then compute and return the required number.

One consequence of using the system described here is that computational programs become truly independent of the machine. Another is that the use of auxiliary storage to complement the fast memory of the computer can be considered as a standard system function. It can be optimized for each machine installation and will in no way alter the structure of the computational programs.

The structure of the computational program is to a large extent determined by the form of the available manipulators. This report describes two major classes of manipulators. These relate to matrix-vector operations and to normal operations involving functions of one variable. In each case an attempt has been made to provide a natural form for the manipulators. It is believed that the form of the computational

programs that use these manipulators will very closely parallel the logical statement of the problem.

Input and output of information is handled by manipulative routines. The form of input is intended to be very flexible and relatively tolerant so that it will accept information in a variety of ways and will make many of the standard conventions unnecessary.

Output in the form of printed information, photographs of an oscilloscope screen, or plots on the off-line analog equipment will all be prepared in a standard form which will perform most of the obvious choices automatically. Additional special-purpose changes have been provided for. While designed primarily for very large programs, it is believed that the use of the programming system implied by this report will greatly ease the task of writing simple programs and will provide untrained programmers with a relatively easy means of obtaining answers to problems.

2. THE ALLOCATOR

All storage of data is considered to be a system function. All the available data-storage area is therefore entrusted to a special set of routines, collectively referred to as the ALLOCATOR. The task of the allocator is to provide efficient use of the storage facilities. It must also be able to communicate information related to this storage to the manipulative routines described in later sections.

Before any specific data can be used, the allocator must receive a request for reservation of space for them. In addition to a reference name for this block of data, descriptive information related to the type of data is supplied with this request. This descriptive information and the location assigned to the data are retained in a special reference list for later use. When a particular named block of data is referred to

by a manipulative routine, this descriptive information is automatically made available for use by that routine.

As requests for storage of information are processed, lists of reserved names and their corresponding descriptive information are constructed. When a particular named reference is no longer needed, the allocator is informed of this and its space is made available for other use.

If sufficient storage is available, the allocator will be able to process all requests. If it becomes necessary for the allocator to move data to provide contiguous space for large requests, this function is performed automatically. The allocator is able to extend its effective storage area by using slower storage media whenever necessary.

Allocator Entries

In the entries for all routines, the arguments have the following meanings.

NAME. This is the alphanumeric name associated with the data. It is normally given in the FORTRAN form `n H xxxx` and may consist of up to 8 characters, the first of which should be alphabetic. For automatic assignment of names, it may be treated as a normal fixed-point variable. (All names will be underlined in this report. This will emphasize the `n H` character of this argument.)

KIND. This is a descriptive designator for the kind of information associated with this name. Its values are not as yet fully set, but the following is a tentative list.

KIND = 0	UNDEFINED
= 1	FIXED-POINT DATA
= 2	FLOATING-POINT DATA

- = 3 DOUBLE-PRECISION DATA*
- = 4 COMPLEX DATA*
- = 5 OCTAL DATA
- = 6 BCD DATA
- = 7-15 SPECIAL CODED DATA

NO DIM. This is the number of dimensions associated with the stored data (limited to at most 2 dimensions).

- NO DIM = 0 A single number
- = 1 A one-dimensional array or a vector
- = 2 A two-dimensional array or a matrix
- = 3 A symmetric matrix stored in upper-diagonal form
- = 4 An antisymmetric matrix stored in upper-diagonal form
- = 5 A function of one variable (1-dim array)

LEN 1. The number of rows in the array (rows for a matrix, components for a vector).

LEN 2. For a matrix, this is the number of columns in the array .

LOC. The location of the first word of the named data in the special array controlled by the allocator.

This section is intended as a descriptive explanation of each of the normal entries to the allocator. The effect of each entry is explained in terms of its function from the viewpoint of the program calling for it.

(a) RESERVE (NAME, KIND, NO DIM, LEN 1, LEN 2, PRESET)

This is the entry which requests the reservation for a new area with the prescribed name. The allocator processes this entry by

* Not yet implemented.

putting the name in its reference tables and assigning to it a specific part of the storage area. This entire area is preset to the value specified by the argument PRESET. If a previous entry with this same name is already present, the old reservation is canceled.

(b) FREE RESV (NAME)

The reservation with this name is removed from the reservation list and its assigned area is made available for other uses.

(c) LOCK RESV

All reservations active at the time of this entry are assigned fixed locations and can no longer be moved by the allocator. Reservations may still be removed from the reference list, but the space is not freed for use by the allocator. Reservations made after this entry are treated normally.

(d) UNLOCK RS

This negates the effect of the LOCK RESV entry. After this entry is processed, the allocator is free to move all reserved areas and will have access to all unreserved spaces.

(e) CLEAR RES

This entry removes from the allocator tables all references not protected by the LOCK RESV. It is normally used at the completion of a major section of a program to remove all temporary reservations.

(f) REDEFINE (NAME, KIND, NO DIM, LEN 1, LEN 2, PRESET)

This entry provides the means of altering the descriptive information associated with a previously defined area. In doing so, the data themselves are automatically made consistent with the changes. All newly created elements are preset to the values indicated.

(g) RENAME (NAME OLD, NAME NEW)

The area previously referred to by the name NAME OLD is now called NAME NEW. If this named area is protected by a LOCK RESV, it will remain protected under this new name. Note the old name can no longer be used for reference to this information.

(h) INF ABOUT (NAME, KIND, NO DIM, LEN 1, LEN 2)

This is a function routine whose value is set equal to the location of the named data, all other arguments are set equal to the information initially assigned to this named reference. This entry point is intended for special-purpose uses in computational programs; an alternative means for accessing this same information is provided for the manipulative routines.

(i) DUMP ALOC

Entering this routine causes a diagnostic dump of the information stored in the allocator reference tables.

(j) N1N DUMP

This gives a dump of all data stored under allocator control. The dump is arranged to provide complete labeling and decoding of all stored information.

(k) DUMP NAME (NAME)

The specified name data are written on the standard output device in the standard form.

(l) N1N RESRV

This is the preset entry which effectively clears and resets all of the allocator functions. It is normally entered prior to the first use of any named-storage routines.

The allocator has additional entry points which are used for special purposes. It should be apparent that additional entries related to normal allocator functions are needed. These will be added to the system

as the need for them arises. The new additions will be of a type that does not interfere with the functions already defined.

3. MANIPULATIVE ROUTINES

Since stored data can be moved by the allocator, it is necessary that each set of data should be indirectly addressed. To do this for each operation would place a serious handicap on the use of this system. For this reason, a large number of so-called manipulative routines are available. These manipulators are coordinated with the allocator and operate directly on stored information. They are controlled in their functions by the computational program and provide the means for such programs to carry out computations efficiently.

For the most part, the manipulative routines operate on sets of data rather than individual numbers. They therefore perform many of the functions which are normally considered to be part of the computational program. Since the descriptive information associated with named data is available to these routines, they are able to perform their functions with a minimum of directive information.

All manipulative routines have standardized features. Most of these are determined by the requirements for interconnection with the other system functions, but some have been added to lessen the possibility of propagating errors in calculation. (The viewpoint taken in constructing the routines is one of safety at the expense of efficiency. For very large programs, it is of utmost importance that the individual routines be fail-safe since it is usually impossible to test the programs under full operating conditions.)

Each manipulator is intended to represent the algorithm for carrying out a specific operation. The calling sequence supplies the names of the data to be operated on and the names for the resultant data. As a

general rule, the manipulators are designed so that space for the resultant information will be reserved if necessary. (In such cases the resultant name is followed by an asterisk in the descriptions which follow.)

It is essential that no large permanent arrays exist in any of the manipulators. (Such arrays are a waste of storage areas whenever the manipulator is not in use.) If temporary areas are needed, they may be requested from the allocator but should be returned when no longer in use.

Although it is possible for individual manipulative routines to carry out computations independently, it is important for efficient operations that the routines be divided into classes that have coordinated functions. It is likely that successive entries to a class of routines will have references to the same named information. With coordinated routines, the references to the allocator can be kept to a minimum.

In the sections that follow, the system manipulators will be described in classes. These routines are those which are available for all uses of the named-storage scheme. It should be clear that before constructing any computational program this class of routines must be augmented with those specific manipulators needed to carry out the special algorithms of the particular computation.

The standard system manipulators are divided into three classes: (1) General Area Manipulators, (2) Elemental Manipulators, and (3) Input-Output Manipulators.

A group of general vector and matrix manipulators are included as part of the system routines because of their frequent use. These are divided into classes designated as: (4) Vector Manipulators, (5) Matrix Manipulators, and (6) Vector-Matrix Manipulators.

A class of manipulators for carrying out simple functional operations is also included in the general manipulative routines of the system as an aid in this type of calculation.

Appendix I contains a list of all the standard entries for the system.

4. GENERAL-AREA MANIPULATORS

(a) COMPARE (NAME 1, NAME 2)

This is a function routine which returns a 0 (FALSE) if the two named sets of data do not agree in all respects. All elements of arrays are compared.

(b) MOVE (NAME 1, NAME 2^{*})

Data under NAME 1 are copied into a new named area called NAME 2. If NAME 2 has not previously been defined or is not able to accept this new set of data, a new reservation is made.

(c) EXCHANGE (NAME 1, NAME 2)

The data referenced as NAME 1 and NAME 2 are exchanged. This is accomplished by exchanging the names in the reference tables.

(d) SET (NAME, VALUE)

This sets all values of the named area to the indicated value. It is used to reset a storage area at some time after the initial reservation.

(e) LIN COMB (COE 1, NAME 1, COE 2, NAME 3, NAME ANSW^{*})

The linear combination of the two named areas is stored in the answer area. If not previously defined, space is reserved for the newly named area. The two areas must agree in kind and dimensionality. The answer area may be identical to either of the input areas if desired.

(f) CUM SUM (COEF, NAME, NAMEANS^{*})

The named area is multiplied by the indicated coefficient and added to the contents of the answer area. If not previously defined, the answer area is reserved and cleared prior to executing this request.

(g) MULT (NAME, COEF)

Each element of the named area is multiplied by the indicated coefficient.

(h) FETCH (NAME, ARRAY)

The named area is copied into an array that is not under allocator control. This is available to provide connections to normal programs with dimension arrays. The size of the space available for ARRAY cannot be checked. The safeguards of the allocator are therefore unable to prevent storage overflows with FETCH operations. The use of FETCH requests should therefore be used with extreme caution.

(i) LOAD (NAME, ARRAY)

An external array is stored in the named storage area. This is the inverse of the operation FETCH. It is necessary that the named area be previously reserved in order that the operations can be carried out.

(j) BIGGEST (NAME, I, J)(k) SMALLEST (NAME, I, J)(l) LARGEST (NAME, I, J)(m) LITTLEST (NAME, I, J)

These functional routines return the extreme values of any array along with the indices that locate the particular elements. Conversion from fixed-point to floating-point form occurs automatically in transmitting this information (e.g., if the request is for the BIGGEST value of some fixed-point area, it will be supplied in floating-point form). It is necessary that the indices be set, even if they are not used in the program. These arguments must therefore be included in the calling statement. Floating-point results are returned by (j) and (k). Fixed-point results are returned by (l) and (m).

5. ELEMENTAL MANIPULATORS

Although it usually is not necessary to operate on individual elements in the stored arrays, such operations are provided for within the general framework of the manipulative routines.

(a) SET VAL (NAME, indices, VAL)

This sets the indicated element to the specified value. (The number of arguments in this entry is dependent on the dimensionality of the named area; e.g., for a vector, a single index must be supplied.)

(b) INCR VAL (NAME, indices, INCREMENT)

(c) ADD TO VAL (NAME, indices, ADDEND)

These two function routines increment the indicated elements. The values of the function are the new value stored in the elemental operation. (b) is a fixed-point entry and (c) is a floating-point entry.

(d) VAL AT (NAME, indices)

(e) INT AT (NAME, indices)

The value of a specific element is returned by these function routines. Both fixed-point and floating-point versions are given here.

6. INPUT-OUTPUT MANIPULATORS

In order to provide for an integrated use of named storage, input and output of data have been provided for within the system. Special manipulative routines are available for such purposes. These are intended to satisfy most input and output requirements. Actually, input control is a major feature of this system and many additional features will be added to the ones given here. The description here should be viewed as a preliminary one.

Printed Output

The printing of stored data can be handled in a very simple manner, provided one of the standard forms of output is considered acceptable. If this is commonly not the case, then the available standard forms should be enlarged. The output routine not only prints large arrays with suitable column and row headings but also puts fixed- and floating-point numbers in a standard legible form. The only extra information needed normally is a heading for the output. To print information in the standard array, one only needs to use the entry:

(a) WRITE OUT (NAME, n H (heading))

The normal carriage control and other facilities of a format are available. The number n must be equal to the number of characters in the format (including both parentheses).

Punched Output

It is intended that facilities for punching out information stored in the named storage areas will be implemented soon. The entries will be

(b) PUNCH N (NAME)

and

(c) PUNCH BIN (NAME)

For each case there will be cards whose heading contains the information in the allocator tables. The cards may then be used as input at a later date. Entry (b) is used to produce cards that can be read and changed easily. Entry (c) puts the information in a more compact form and should be favored for general output.

Output to Tape

In order to be a self-contained and complete system, there are provisions for writing data on tapes. Since such information can only be intended for use by this system, the tapes are written in the binary mode without internal conversion. Each set of data written out is given a heading containing the information in the allocator reference for this set. The entry is

(d) WRITE TP (NAME, NO OF TAPE)

Input

Input directly to the allocator has many virtues. For this reason most input will be read in by standard manipulative routines. Such information can either be in the forms produced by the routines described above or in a free-field form for manually-produced data. For entering information in standard form, a heading that contains the descriptive information usually precedes each block of data.

The entries that follow may have a terminal X added to indicate that the operation refers to the next input and that the name will be read in. Without this suffix, the name of the data is compared with the argument in the call, and successive blocks will be read until the referenced set of data is located.

(e) INPUT (NAME) or

INPUT X(NAME)

Information produced by (b) or (c) above can be entered into the

allocator by this entry. Manually-produced data will also be accepted.

(f) READ TP (NAME, NO OF TAPE) or

READ TPX (NAME^{*}, NO OF TAPE)

This is the inverse of operation (d). In each of these cases the information read is entered directly into the allocator. If the named area is not previously defined or cannot accept this information, a new reservation will be made.

Graphs

Graphical information can be obtained directly from the stored data. At present only a simple plotting routine exists, but more will be added in the future.

(g) GRAPH (NAME 1, NAME 2)

A graphical display of pairs of points in the two arrays is photographed. The first-named array is the vertical control and the second the horizontal control. If either of the areas is a 2-dimensional array, several curves are produced simultaneously. The scales and labels are automatically included in a way that is intended to satisfy most users.

(h) DRAW (NAME 1, NAME 2)

A hard copy of the display described above is produced on the CALCOMP plotter.

7. MATRIX AND VECTOR MANIPULATORS

This section describes the standard manipulative routines for matrix and vector operations. These elements of the system may also be considered as an example of a class of manipulators that might be constructed for the specific needs of a calculation. It should be noted that these routines are intended to be a complete set of operations. If any of the required matrix or vector operations cannot be constructed from the following, additional entries are needed.

A. Vector Handlers

- (a) DOT VECS (NAME 1, NAME 2) or DOT PROD (NAME 1, NAME 2)

This function routine computes the inner product of two previously defined vectors. These vectors must be of the same length.

- (b) VEC NORM (NAME)

The vector is normalized to unity by this operation. The length of the vector prior to normalization is returned as the functional value.

- (c) VEC LENGT(NAME)

This function routine returns the length of the named vector.

B. General Matrix Handlers

- (a) XPOSE MAT (NAME)

This routine transposes the rows and columns of a matrix.

- (b) DET OF MAT (NAME)

The determinant of a named square matrix is returned as the value of this function routine.

- (c) DIAG MAT (NAME, NAMEVECS^{*}, NAMEVALS^{*})

This routine is used to obtain the eigenvalues and eigenvectors of a square matrix. The eigenvectors and eigenvalues are stored in the specified named areas, which will be assigned if not previously reserved. The results are arranged in the order of increasing eigenvalue; the eigenvectors are the corresponding rows of the resultant matrix. The trace of the matrix is returned as the functional value of this routine.

- (d) DIAG S MAT (NAME, NAMEVECS^{*}, NAMEVALS^{*})

For a symmetric matrix, a routine which is faster than (c) is available. Its arguments and functions are identical to that routine.

(e) RECIP MAT (NAME, NAMEANS^{*})

This routine inverts the indicated square matrix and returns the determinant of the matrix as its functional value.

(f) MAT BY MAT (NAME 1, NAME 2, NAMEANS^{*})

The two matrices are multiplied together and the new product matrix is stored in the assigned area. The number of columns in the first matrix must agree in number with the number of rows of the second.

(g) ROW TO ROW (NAME 1, NO ROW 1, NAME 2, NO ROW 2)

This and the following three routines are used to transfer individual rows or columns from one matrix to another.

(h) COL TO COL (NAME 1, NO COL 1, NAME 2, NO COL 2)(i) ROW TO COL (NAME 1, NO ROW 1, NAME 2, NO COL 2)(j) COL TO ROW (NAME 1, NO COL 1, NAME 2, NO ROW 2)

C. Matrix Row Operations

(a) LOAD ROW (NAME, I, EXTVEC)

The indicated vector is stored in the Ith row of the named matrix. Note that this routine communicates with data that are not part of the information under allocator control. The argument EXTVEC is the first location of the information to be transmitted to the allocator area.

(b) FETCH ROW (NAME, I, EXTVEC)

This is the inverse of (a). These two routines provide the means of adapting parts of normal programs to the named-storage routines. (See comments for routine FETCH, Section 3.)

(c) ROW NORM (NAME, I)

The Ith row of the named matrix is normalized.

(d) ROW LENGT (NAME, I)

The length of the Ith row of the named matrix is returned by this function routine.

(e) L COM ROWS (NAME, COEI, I, COEJ, J, K)

The linear combination of COEI times the Ith row and COEJ times the Jth row is placed in the Kth row. Row K may be the same as I or J.

(f) DOT ROWS (NAME, I, J)

This function routine returns the inner product of the Ith row with the Jth row of the named matrix.

(g) EXCH ROWS (NAME, I, J)

The Ith and Jth rows of the named matrix are exchanged.

(h) VEC TO ROW, (NAME VEC, NAME MAT, I)

The named vector is copied onto the Ith row of the named matrix.

(i) ROW TO VEC (NAME MAT, I, NAME VEC*)

The Ith row of the named matrix is copied into the named vector.

(j) ADD V TO RW (NAME VEC, NAME MAT, I)

The named vector is added to the Ith row of the named matrix.

D. Matrix Column Operations

These are identical to those listed for the row operations. In each case replace ROW by COL in the calling forms. The corresponding operation is then defined for the columns of the matrix. Row operations ordinarily are more efficiently handled than column operations. For this reason, whenever possible, assignments should be made with this feature in mind.

E. Vector-Matrix Operations

(a) MAT BY VEC (NAME MAT, NAME VEC, NAMEANS*)

The matrix-vector multiplication is performed. The resultant vector is stored in the specified area which can be reserved if necessary by this routine.

(b) VEC BY MAT (NAME VEC, NAME MAT, NAME ANS^{*})

The named vector is treated as a row matrix and a matrix multiplication is performed. The resultant area is set to correspond to a column vector.

(c) G DOT VECS (NAME VEC 1, NAME MAT, NAME VEC 2) or
EXPEC VAL (NAME VEC 1, NAME MAT, NAME VEC 2)

This functional routine returns the generalized dot product of two vectors with a square matrix corresponding to the metric of the space. It also corresponds to the evaluation of the expectation value of an operator in the quantum-mechanical formulation of a problem.

8. FUNCTION MANIPULATORS

It should be clear that the manipulators described thus far are for the most part oriented towards problems that involve vector and matrix operations. For many problems, these types of operations are not the ones most frequently used. The ideas related to the construction and interconnection of manipulative routines are, however, easily adapted to other classes of problems.

Here we will describe a second class of manipulators that have been constructed to help in the applications of this system to problems that involve functions of one variable and in which functions are evaluated at specific grid points.

The independent variables of any problem can be defined in a way that specifies not only the range of this variable but also the values of the variable at each grid point. This can, of course, be done in many ways. A simple form is given here, with the understanding that others will probably be added. This form produces an equally-spaced mesh.

(a) EQUI GRID (NAME, VAL 1, VAL 2, DELTA)

This manipulator will form an equally-spaced grid for the interval between VAL 1 and VAL 2 with a grid spacing of DELTA.

Unary operations such as those performed in FORTRAN operations can be carried out for all values of a function by simple manipulators. The form of all unary manipulators is fixed and are in a one-to-one correspondence with the standard FORTRAN functions. Binary operations relating to the functions are performed with additional manipulators related to basic arithmetic operations. Examples of these operations are the following.

<u>FORTRAN</u>			<u>MANIPULATOR</u>	
ANS = XXX F (ARG)			CALL FUN XXX (NAME ARG, NAME ANS)	
<u>Unary</u>				
(b)	FUN	SIN	(<u>NAME ARG 1,</u>	<u>NAME ANS[*]</u>)
(c)	FUN	COS	("	")
(d)	FUN	TAN	("	")
(e)	FUN	LOG	("	")
(f)	FUN	EXP	("	")
(g)	FUN	ABS	("	")
(h)	FUN	SQRT	("	")
(i)	FUN	SQ	("	")
<u>Binary</u>				
(j)	FUN	MULT	(NAME ARG 1, NAME ARG 2, NAME ANS [*])	
(k)	FUN	DIV	("	" ")
(l)	FUN	ADD	("	" ")
(m)	FUN	SUB	("	" ")

Note that these manipulator operations relate to the functions on a point-by-point basis. Thus MULT and FUN MULT perform entirely different operations: MULT multiplies every value of an area by the same value, while FUN MULT multiplies each value by some other value which can vary from point to point.

For most functional calculations it is necessary to carry out differentiation and integration. Several forms of such routines will be available eventually. For the moment, simple numerical integration and differentiation schemes have been supplied. They are:

- (n) QUADRATE (NAME FUNC, NAME VAR, NAME ANS*)
- (o) DIFFEREN (NAME FUNC, NAME VAR, NAME ANS*)

For (n), the functional value is the total integral.

To ascertain the value of a function at any specified value of the independent variable, it is necessary to have a manipulator that can interpolate between points. Again this is done by simple numerical methods.

- (p) FUN.VALUE (NAMEFUN, NAMEVAR, VAL VAR)

9. PSEUDO ARRAYS

It is often desirable to have functional routines with only index arguments. If one is working with a particular area frequently and would like to reference it in a more natural way, special functions can be constructed to address specific named areas without that name appearing as an argument. For example,

AJ = HOKUM (I)

could reference the Ith element of some named area. The function routine named HOKUM could act as an ordinary manipulator but with the built-in named references. Any one- or two-dimensional area could be addressed in this way. Additional entries would be needed to set and alter the elements of this area. For example,

SET HOKUM (I, VALUE)

and

ADD HOKUM (I, ADDEND)

For each array used in this way, special routines would be needed.

Two routines of a similar type are available as part of the system. These do not have specific names attached to the reference. The names are set by special entries. These routines and their entries are:

(a) SET E NAME (NAME*, NO DIM, LEN 1, LEN 2, PRESET)

Set the name to be used for subsequent references to this routine.

(b) ELEMENT (I)

Return the value of the Ith element. (If this is a 2-dimensional array, replace I by I, J.)

(c) SET ELEMT (I, VALUE)

Set the designated element.

(d) ADD TO ELE (I, VALUE)

Add to the designated element and return the new value. A similar set of routines is available for use with fixed-point areas.

(e) SET I NAME (NAME*, NO DIM, LEN 1, LEN 2, PRESET)

(f) INTEGER (I)

(g) SET INTEG (I, INTE)

(h) INCR INTG (I, INCR)

10. REFERENCING AND ERROR DETECTION

All manipulative routines should perform tests to detect possible error conditions. To a large extent these tests can be performed automatically by the proper use of the scheme for communication with the allocator. Manipulative routines request information about specific named data by calling a special routine NAMEDATA. All arguments are transmitted

via a special labeled common region. This common region contains not only the name of the data and all of the descriptive information but several key words that are used to selectively test the values of any of the descriptive information and to interrupt the computation if specified conditions are not met. The controlling words are set by the calling program and tested by the allocator routine that supplies the descriptive information.

The descriptive information for a specific named reference is requested by setting NAME to be the name of the area. After executing the request the following arguments are set:

- (1) LOC the location of the named area
- (2) KIND the kind of information in this area
- (3) NO DIM the number of dimensions
- (4) LEN 1 the length in direction 1
- (5) LEN 2 the length in direction 2

Each argument also serves as a test control element which can be set prior to the request.

If tests are to be performed, the special control word NAME TEST must be set prior to the request to indicate the particular tests desired. The code used to control these tests is as follows:

- 0 No tests are to be performed.
- 2 Is KIND = (KIND)
- 4 Is NO DIM = (NO DIM)
- 8 Is LEN 1 = (LEN 1)
- 16 Is LEN 2 = (LEN 2)
- 32 Is NO DIM $\geq$ (NO DIM)
- 64 Is LEN 1 $\geq$ (LEN 1)
- 128 Is LEN 2 $\geq$ (LEN 2)

Here () indicates the preset value of this quantity. Any combination of the above tests can be requested by setting the sum of the corresponding numbers as the value of NAME TEST. If the test-element value 1 is present in this sum, the information is being used as a resultant. If it has not previously been defined in a way that is compatible with the selected tests, it will be reserved. NAME TEST is always reset to zero before returning to the calling program.

The consequence of failing a test may be controlled by the value of the word NAME FAIL. If the value is zero, the calculation is terminated with a diagnostic dump. If it is +1, the computation continues but NAME FAIL is set to -N (where N is the code given in the list above) to indicate the type of failure to the calling program. If NAME FAIL is not reset before the next use of this routine, an unconditional interrupt will occur.

An example of the use of this communication and test routine in a manipulative routine might be as follows:

```
NAME TEST = 70      (64 + 4 + 2)
KIND = 2
LEN 1 = 3
NO DIM = 1
NAME = 3HVEC
CALL NAME DATA    (TEST: KIND = 2, LEN 1 ≥ 3
                   and NO DIM = 1)
```

If VEC exists, is of KIND 2, and is a vector of length greater than 2, then the common area will have all of the descriptive data set. Otherwise the computation will have been interrupted.

Additional tests performed within a manipulative routine may indicate an error. In such cases the standard diagnostic termination may be requested by calling N1NERROR. All information relating to the previous named request will be given as part of this termination.

11. NAMING LOCAL AREAS

In many cases areas are defined for uses which bear on the over-all calculation only in an indirect way. For subprograms in particular, it is often necessary to make reservations whose meanings are either local to the specific routine or to other connected routines. It is essential that the names chosen for such areas be distinct from those used in unrelated routines. The allocator is provided with an automatic naming facility which can be used to provide such distinct names automatically. This facility also makes it particularly easy to release such temporary storage areas at the completion of their local use.

The facility relies on a property of the FOR TRAN-63 compiler which is only partially implemented at the present time, but which can be artificially effected by temporary additions to present programs. The specific property used is that any variable is initially zero unless it is preset by a DATA statement. The temporary addition is such a DATA statement that presets the names to zero.

The allocator has been designed to interrogate all requests to determine if a proper name has been transmitted (zero is not a proper name). If this is not the case, a unique name is defined and replaces the value of the calling argument. In this way an automatic naming convention is provided and the integrity of separate routines is maintained. The advantage of this method of naming areas is that temporary storage is kept under direct control and it is possible to define areas to have local meanings. The particular form in which this is done also makes the structure of the program more natural; the nHxxxx method of naming information is not used.

The major disadvantage is that the allocator name for a specific area bears no relationship to the actual variable name used in the program and a diagnostic dump will contain information with names that

differ from those in the program. An example of this use might be as follows. Consider a subroutine which is intended to print a certain linear combination of two vectors. It could be written in the form:

```
SUBROUTINE SOMEONE (VEC 1, VEC 2)
DP = DOT VECS (VEC 1, VEC 2)
CALL LIN COMB (-DP, VEC 1, 1.0, VEC 2, VEC 3)
CALL WRITE OUT (VEC 3, 11 H, (*ANSWER*))
CALL FREE RESV (VEC 3)
END.
```

Note here that VEC 3 was an implied resultant and had no previous value assigned to it. It was given a name automatically by the allocator and the program has used this name for other uses; the particular form of that name is irrelevant.

In order to provide for the release of many temporary storage areas assigned in this way, it was necessary to define a pair of routines that can be used to simultaneously free all locally-defined areas. These are used with the following statements:

- (a) CALL LOCAL BEG
- (b) CALL LOCAL END.

These statements must appear in pairs in any program and will serve to automatically clear any local reservations which have appeared between them. Normally these should be the first and last statements in any subprogram.

Local areas are defined as local from above, i.e., they may be used in a normal manner by any sub-subprograms called within a subprogram, provided their names have been transmitted by argument to that subprogram. Local definitions can also be made within the lower level subprograms and will be properly treated. This is true to a depth of 20 levels.

The use of local arrays constitutes one of the most important features of the use of the named-storage concept since it almost automatically eliminates the use of storage for dead data.

12. EXAMPLE OF A MANIPULATOR

As an illustrative example of the form of a manipulative routine, a FORTRAN version of DOT ROWS is presented here. Normally a multiple-entry routine would be used to coordinate several members of the same class of manipulators. Here the allocator is assumed to have a 16k region available for its storage area.

```
FUNCTION DOT ROWS (NAMEMAT, NROW1, NROW2)
```

```
COMMON /NAMEDA /NAME, LOC, KIND, NODIM, LEN1, LEN2,
```

```
KEYM, NAMETEST, NAMEFAIL
```

```
COMMON/ALLOCSTO/NO PACKS, ALLOC(16000)
```

```
    NAMETEST      = 70
```

```
    KIND          = 2
```

```
    NODIM         = 2
```

```
    LEN 1        = XMAXOF (NROW1, NROW1)
```

```
    NAME          = NAMEMAT
```

```
    CALL NAME DATA
```

```
    LOCR 1       = LOC + (NROW1 -1) *LEN 1 -1
```

```
    LOCR 2       = LOC + (NROW2 -1) *LEN 1 -1
```

```
    DOT ROWS     = 0.0
```

```
    DO 1 I       = 1, LEN 1
```

```
1    DOT ROWS    = DOT ROWS + ALLOC(I+ LOCR1)*ALLOC(I+ LOCR2)
```

```
    END
```

13. CONCLUDING REMARKS

The presentation in this report is primarily descriptive. The advantages of the use of named storage have for the most part been omitted. This concluding section is intended to compensate for this by overemphasis. In this way the reader is perhaps left with less of the details and more of the over-all features of this approach.

Named storage is a data-storage scheme which replaces those features of FORTRAN that are not concerned with FORmula TRANslation but which have been added to the language to direct the storage-allocation features of the compiler. Statements such as DIMENSION or COMMON are not really statements that belong to a formulation of the mathematical problem.

The fact that the parameters used in these statements must be carefully considered when dealing with large problems is indeed to be considered a strange feature in the structure of a computer language. It asks a person to decide how to allocate the various parts of the memory of the computer, even though it is a modern computer that is capable of carrying out such trivial bookkeeping problems with ease.

The use of named storage is then a method of constructing a sublanguage that eliminates the "un-FORTRAN" statements from FORTRAN. In doing this, much of the trivial bookkeeping that a programmer normally performs is assigned to the computer as a subsidiary task in the mathematical operations called for by the computational program.

Many of the difficulties encountered in constructing programs are effectively nonexistent with this method of data storage. It is not possible for information to stray outside of the predefined bounds assigned to particular named data. If abnormal requests are encountered, the system automatically terminates the program with a full diagnostic dump.

This effective "bounds-control" feature is one whose value cannot be overemphasized, not only because it eliminates error propagations almost immediately, but because it provides a means for memory sharing by logically unrelated programs. If all storage and manipulation of data is performed by standard manipulators, it is not possible for the two programs to interact. Each is limited to operating only in its assigned data areas.

One particular use of the named-storage concept is for on-line time-shared programs on a machine such as the CDC-3600. Not only does this technique provide the basis for a compact directive form necessary for easy communication; it also enables the user to have access to all stored information by simple named references. A more important aspect is the fact that it is possible to use the memory jointly with another program without setting aside a specific amount of memory for either program. The amount of memory used by the on-line program can vary to match its current requirements and the rest of the memory can always be available to the operating program. Request for new storage for the on-line program can be made in increments and can be accepted when the other program is able to conveniently supply the needed area. In this way the over-all use of storage space can be optimized dynamically by the computer itself.

A cursory reading of this report might imply that one has for the most part replaced all dimension statements by corresponding statements calling for reservations. This is far from true. Very few reservation requests have to be made directly. All manipulative routines that create new results can automatically make reservations and perform the necessary bookkeeping associated with these requests. The allocator functions are therefore for the most part subsurface and its use is not apparent in the structure of the computational program.

As a concluding paragraph, consider the following small program used to orthonormalize any number of vectors whose lengths are specified at input time.

```

PROGRAM SCHMIDT
C   READ A SET OF VECTORS, PRODUCE ORTHONORMAL SET

NAME = 8HVECTOR00
I = 0
1   IF (INPUT X(VEC) ) 2,5
2   ITH = NAME + I + 1
    CALL RENAME(VEC, ITH)
    IF (VEC NORM (ITH) )3,1
3   DO 4 K = 1, I
    DP = DOT VECS (ITH, NAME + K)
4   CALL CUM SUM (-DP, NAME + K, ITH)
    IF (VEC NORM (ITH) ) I = I + 1
    GO TO 1
5   DO 6 K = 1, I
6   CALL DUMP NAME (NAME + K)
END

```


APPENDIX : LIST OF STANDARD ENTRIES

The following is a list of the standard entries that are now available. More entries will undoubtedly be added in the future.

Allocator Entries

RESERVE	REDEFINE	N 1 N RESRV
FREE RESV	RENAME	NAMEDATA
LOCK RESV	DUMP ALOC	N 1 N ERROR
UNLOCK RS	N 1 N DUMP	INF ABOUT
CLEAR RES	DUMP NAME	

General-Area Entries

COMPARE	CUM SUM	BIGGEST
MOVE	MULT	SMALLEST
EXCHANGE	FETCH	LARGEST
SET	LOAD	LITTLEST
LINCOMB		

Elemental-Operation Entries

SET VAL	ADD TO VAL	VAL AT
	INCR VAL	INT AT

Input-Output Entries

WRITE OUT	WRITE TP	GRAPH
PUNCHN	READ TP	DRAW
PUNCH BIN	INPUT	

Vector Entries

DOT VECs

VEC NORM

VEC LENGT

Matrix Entries

XPOSE MAT

ROW TO ROW

DIAG MAT

RECIP MAT

COL TO COL

DIAGS MAT

DET OF MAT

ROW TO COL

MAT BY MAT

COL TO ROW

Row Entries

LOAD ROW

L COMB RWS

VEC TO ROW

FETCH ROW

DOT ROW

ROW TO VEC

ROW NORM

EXCH ROWS

ADD V TO RW

ROW LENGT

Column Entries

LOAD COL

L COMB CLS

VEC TO COL

FETCH COL

DOT COLS

COL TO VEC

COL NORM

EXCH COLS

ADD V TO CL

COL LENGT

Vector Matrix Entries

VEC BY MAT

MAT BY VEC

G DOT VECs

Function Entries

EQUI GRID

FUN MULT

FUN DIV

FUN ADD

FUN SUB

FUN SIN

FUN COS

FUN TAN

FUN LOG

FUN EXP

FUN ABS

FUN SQRT

FUN SQ

FUN VALUE

QUADRATE

DIFFEREN

Pseudo-Array Entries

SET E NAME

ELEMENT

SET ELEMT

ADD TO ELE

SET I NAME

INTEGER

SET INTEG

INCR. INTG