

ESD TDR 64-390

ESTI FILE COPY

Technical Documentary  
Report No. ESD-TDR-64-390

ESTI PROCESSED

DDC TAB     PROJ OFFICER

ACCESSION MASTER FILE

\_\_\_\_\_

DATE \_\_\_\_\_

ESTI CONTROL NR. DL-41 190

CY NR 1 OF 1 CYS

**ESD RECORD COPY**

RETURN TO  
SCIENTIFIC & TECHNICAL INFORMATION DIVISION  
(ESTI), BUILDING 1211

COPY NR. \_\_\_\_\_ OF \_\_\_\_\_ COPIES

**MILITRAN  
REFERENCE MANUAL**

Prepared for the

OFFICE OF NAVAL RESEARCH  
NAVY DEPARTMENT  
WASHINGTON, D. C.

and the

DIRECTORATE OF COMPUTERS  
ELECTRONIC SYSTEMS DIVISION  
AIR FORCE SYSTEMS COMMAND  
L. G. HANSCOM FIELD, BEDFORD, MASS.

U. S. Navy Contract No. Nonr 2936(00)

by

SYSTEMS RESEARCH GROUP, INC.  
1501 Franklin Avenue  
Mineola, L. I., New York

JUNE 1964

ADD 601794

When US Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Reproduction in whole or in part is permitted for any purpose of the United States Government.

DDC AVAILABILITY NOTICE

Qualified requesters may obtain copies of this report from the Defense Documentation Center (DDC), Cameron Station, Alexandria, Va. 22314. Orders will be expedited if placed through the librarian or other person designated to request documents from DDC.

# **MILITRAN REFERENCE MANUAL**

Prepared for the

**OFFICE OF NAVAL RESEARCH  
NAVY DEPARTMENT  
WASHINGTON, D. C.**

and the

**DIRECTORATE OF COMPUTERS  
ELECTRONIC SYSTEMS DIVISION  
AIR FORCE SYSTEMS COMMAND  
L. G. HANSCOM FIELD, BEDFORD, MASS.**

**U. S. Navy Contract No. Nonr 2936(00)**

by

 **SYSTEMS RESEARCH GROUP, INC.**  
1501 Franklin Avenue  
Mineola, L. I., New York

**JUNE 1964**

When US Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Reproduction in whole or in part is permitted for any purpose of the United States Government.

#### DDC AVAILABILITY NOTICE

Qualified requesters may obtain copies of this report from the Defense Documentation Center (DDC), Cameron Station, Alexandria, Va. 22314. Orders will be expedited if placed through the librarian or other person designated to request documents from DDC.

## FOREWORD

This is one of three technical reports being published simultaneously. The others are the MILITRAN Operations Manual for IBM 7090-7094 (Technical Documentary Report No. ESD-TDR-64-389) and the MILITRAN Programming Manual (Technical Documentary Report No. ESD-TDR-64-320). The three reports constitute a complete description and instructions for using the MILITRAN language in computer programming of simulation problems.

The MILITRAN 7090-7094 Processor, which is used to compile a problem written in MILITRAN source language into a machine language program, will be available to prospective users. Pending final arrangements, requests for information about the MILITRAN Processor should be sent to the Office of Naval Research (Code 491).

This report was prepared by the Systems Research Group, Inc., under Contract Nonr-2936(00), which was initiated by the Naval Analysis Group, Office of Naval Research, and has been jointly supported by the Office of Naval Research and the Electronic Systems Division, Air Force Systems Command.


## ABSTRACT

MILITRAN is an algorithmic computer language specifically oriented to the problems encountered in simulation programming. In addition to providing overall flexibility in expressing complex procedures, the language contains features which greatly simplify the maintenance of status lists, handling of numeric and non-numeric data, and sequencing of events in simulated time.

This report is intended as a reference summary for those already familiar with MILITRAN.

## REVIEW AND APPROVAL

This Technical Documentary Report has been reviewed by the Electronic Systems Division, U. S. Air Force Systems Command, and is approved for general distribution.

  
J. B. CURTIS  
2nd Lt., USAF  
PROJECT OFFICER



## TABLE OF CONTENTS

	Page
INTRODUCTION	1
GENERAL LANGUAGE CHARACTERISTICS	2
Characters	3
Names	4
Constants	5
Punctuation Marks	7
Operators	7
Mnemonic Delimiters	9
Statement Type Identifiers	9
STRUCTURE	12
Object	12
Class	13
Real, Integer, Logical, Program Object	14
Normal Mode	15
Vector, List	16
Contingent Event, Permanent Event	17
Common	18
Procedure	20
EXPRESSIONS	22
Expression Syntax	22
Functions	25
Standard Functions	26
Subscripts	29
PROCESSING	31
Substitution	31
List Processing	32
List Entries	32
List Processing Conditions	33
Place, Place Entry	35
Remove, Remove Entry	36
Replace, Replace Entry	37
Reset Length	38
List Entry Locating Functions	39
INPUT/OUTPUT	40
Logical Unit Designations	40
Tape Control Statements	40
Input/Output Lists	42

	Page
Binary Read, Binary Write	43
Read, Write	44
Format	45
CONTROL	51
Go To	51
If, Unless	51
Pause, Stop	53
Execute, Return	53
Do, Continue	55
End, Next Event, Next Event Except	57
End Contingent Events, End File Return, End Record Return	60
COMPILER	62
End Compilation	62
Suspend FAP Listing	62
APPENDIX	63
INDEX	68



## INTRODUCTION

This manual is designed as a reference for programmers working in MILITRAN. The description of the MILITRAN language herein is more concise than that given in the MILITRAN PROGRAMMING MANUAL, with which the reader is assumed to be familiar.

Various sections of this manual outline GENERAL LANGUAGE CHARACTERISTICS, statements which define program STRUCTURE, characteristics of EXPRESSIONS, and statements whose major functions include PROCESSING, INPUT OUTPUT, CONTROL, and operation of the COMPILER.

A summary of all MILITRAN forms is included in the appendix.

## GENERAL LANGUAGE CHARACTERISTICS

A MILITRAN source program is a series of MILITRAN statements which specify a sequence of operations to be performed by a digital computer. A program is either a main program or a procedure. A main program initiates processing and may be devised in such a way as to require no procedures. A procedure cannot initiate processing and must receive a signal from either a main program or another procedure before its operation sequence can be executed.

A MILITRAN statement is a string of elements arranged in a prescribed order which specifies one or more of the following characteristics of the program:

1. STRUCTURE of the program  
or its elements;
2. PROCESSING to be performed  
within the computer;
3. INPUT/OUTPUT, or exchange  
of data between the computer  
and its external storage devices;
4. CONTROL of the sequence in  
which various operations are  
to be performed; and

5. COMPILER instructions, or specification of the manner in which the translation from MILITRAN to machine language is to be performed.

The elements which combine to form MILITRAN statements are names, constants, punctuation marks, statement type identifiers, operators, and mnemonic delimiters. These elements are in turn made up of characters, which are the basic units of any language.

#### Characters

The MILITRAN Basic Language is expressed in terms of the following character set:

ABCDEFGHIJKLMNØPQRSTUVWXYZ  
XYZ 0123456789.( ), = + - \* /

The character "blank" is normally not significant in the language. Except where specifically noted in this summary, blanks may be used in any part of a statement without any effect on the statement.

"Alphabetic characters" include the letters A through Z; "numeric characters" include the digits 0 through 9; alphanumeric characters include both alphabetic and numeric characters. All others are "special characters."

## Names

A name is a string of one to sixty alphameric characters, the first of which is alphabetic. Although statement type identifiers and mnemonic delimiters are alphabetic strings, their use within a statement distinguishes them from names without ambiguity.

Certain names have a pre-defined meaning in MILITRAN and may be used only in reference to that meaning.

These names are:

ABS	GST	PRINTER
ATAN	INDEX	RANDØM
ATTACKER	INTEGER	RANDØM INDEX
CARDS	LENGTH	REAL
CØS	LØG	SIGN
EACH	LST	SIN
END CØMPILATIØN	MAX	SQRT
EPSILØN	MIN	TAN
EXP	MINIMUM INDEX	TARGET
FALSE	MØD	TIME
FØRMAT	NEXT EVENT	TRUE

All names used in a MILITRAN source program are either explicitly or implicitly assigned a type. Some types of names are assigned a mode. The type of a name

indicates the nature of its use in the program. The mode of a name indicates the form of data referred to by the name.

The following table lists all possible types of names, whether or not they have modes, and short descriptions of their use in a program.

<u>Type</u>	<u>Mode?</u>	<u>Use in Program</u>
Single Variable	Yes	Storage of single items of data.
Array	Yes	Storage of several items of data under a single name.
Vector	Yes	Storage of several arrays under a single name.
List	Yes	Special form of vector which permits automatic updating of data items.
Object	No	Specification of basic identifiers.
Class	No	Grouping of objects.
Contingent Event with List	Yes	Association of processing functions with a list of the same name.
Permanent Event	No	Linking of processing functions in a simulated time sequence.
Permanent Event with List	Yes	Association of processing functions with a list of the same name.
Vector Component	Yes	An array which is associated with a vector of list.
Symbolic Dimension	Yes	Specification of array dimensions.
Statement Label	No	Designation of points in program.
Procedure	Yes	Designation of subroutine entry.
Open Procedure	Yes	Designation of integral processing codes.
External Procedure	Yes	Designation of separately coded processing.

## Constants

Constants are single items of data whose value is unchanged throughout the execution of the program. In fact, a constant might be thought of as a nameless single variable.

Integer constants may take one of the following forms:

1. A string of numeric characters.
2. A string of the form "nHxxxxxx" where n is a digit not exceeding 6 and x is any character including the character "blank". The number of characters (x) must be equal to n.

Real constants may take one of the following forms:

1. A string of numeric characters which includes one and only one period.
2. A string of numeric characters, which may or may not include a period, followed by E, En, Enn, E+n, or E+nn, where n is a numeric character.

The distinction between real and integer constants is significant only in arguments to procedures.

Logical constants include only the names TRUE and FALSE.



### Punctuation Marks

The only punctuation marks used in MILITRAN are the following:

- . Period
- ( Open Parentheses
- ) Close Parentheses
- , Comma
- ... Ellipsis (Delimits comments)

### Operators

The operators used in MILITRAN are the following:

- = Substitution
- + Addition; plus
- Subtraction; minus
- \* Multiplication
- / Division
- .P. Exponentiation
- .E. Comparison: Equal to
- .G. Comparison: Greater than
- .L. Comparison: Less than
- .NE. Comparison: Not equal to
- .GE. Comparison: Greater than or equal to
- .LE. Comparison: Less than or equal to
- .IS. Object identity
- .IN. Object inclusion



.ØR. Logical disjunction  
.NØT. Logical negation  
.AND. Logical conjunction  
.EQV. Logical equivalence  
.EXØR. Logical exclusive disjunction

### Mnemonic Delimiters

The delimiters BY, BY ENTRY, CØNTAINS, FØR, FRØM, IN, TØ, and UNTIL are used within certain statements. Use of these alphameric combinations as names is permitted, as the distinction between name and delimiter is always contextually clear.

### Statement Type Identifiers

The basic statement in MILITRAN involves substitution of one data item for another within the computer. The substitution statement has the form

$$a = b$$

where a is a subscripted or unsubscripted variable name and b is any expression whose value is suitable for storage in a.

All statements which are not substitution statements are designated by alphabetic strings called statement type identifiers. The following table lists all statement types and their primary uses. The form and characteristics of each statement is summarized in later sections by primary use. Primary functions are listed under GENERAL LANGUAGE CHARACTERISTICS.

<u>Statement Type</u>	<u>Primary Function</u>
BACKSPACE	INPUT/OUTPUT
BACKSPACE FILE	INPUT/OUTPUT
BINARY READ	INPUT/OUTPUT
BINARY WRITE	INPUT/OUTPUT
CLASS	STRUCTURE
CØMMØN	STRUCTURE
CØNTINGENT EVENT	STRUCTURE
CØNTINUE	CONTROL
DØ	CONTROL
END	CONTROL
END CØMPILATIØN	COMPILER
END CØNTINGENT EVENTS	CONTROL
END FILE	INPUT/OUTPUT
END FILE RETURN	CONTROL
END RECØRD RETURN	CONTROL
EXECUTE	CONTROL
FØRMAT	INPUT/OUTPUT
GØ TØ	CONTROL
IF	CONTROL
INTEGER	STRUCTURE
LIST	STRUCTURE
LØGICAL	STRUCTURE
NEXT EVENT	CONTROL
NEXT EVENT EXCEPT	CONTROL

<u>Statement Type</u>	<u>Primary Function</u>
NORMAL MØDE	STRUCTURE
ØBJECT	STRUCTURE
PAUSE	CONTROL
PERMANENT EVENT	STRUCTURE
PLACE	PROCESSING
PLACE ENTRY	PROCESSING
PRØCEDURE	STRUCTURE
PRØGRAM ØBJECT	STRUCTURE
READ	INPUT/OUTPUT
READWRITE	INPUT/OUTPUT
REAL	STRUCTURE
REMØVE	PROCESSING
REMØVE ENTRY	PROCESSING
REPLACE	PROCESSING
REPLACE ENTRY	PROCESSING
RESET LENGTH	PROCESSING
RETURN	CONTROL
REWIND	INPUT/OUTPUT
STØP	CONTROL
SUSPEND FAP LISTING	COMPILER
UNLESS	CONTROL
UNLØAD	INPUT/OUTPUT
VECTØR	STRUCTURE
WRITE	INPUT/OUTPUT

STRUCTURE

Structure - defining statements are summarized in this section. They include:

CLASS	OBJECT
COMMON	PERMANENT EVENT
CONTINGENT EVENT	PROCEDURE
INTEGER	PROGRAM OBJECT
LIST	REAL
LOGICAL	VECTOR
NORMAL MODE	

Object

The statement

$$\text{OBJECT } n_1(d_1), n_2(d_2), \dots, n_m(d_m)$$

defines names  $n_1, n_2, \dots, n_m$  to represent basic object types. The names are preserved for use at running time in input/output operations.

Dimension  $d_1$  designates the number of objects to be named  $n_1$ . This dimension may be an expression of real or integer mode. Names used in a dimension are defined by such use to be symbolic dimensions, and no other declaration of type or mode is permitted except subsequent use in dimensions.

Class

The statement

CLASS (n) CØNTAINS  $a_1, a_2, \dots, a_m$

defines the name n to be that of a class. The name is not preserved in its external form.

The members of the class are specified by elements  $a_1, a_2, \dots, a_m$ , where  $a_1$  may have the following forms:

object name

EACH\*object name

class name

EACH\*class name

Object and class names used in  $a_1, a_2, \dots, a_m$  must have been declared as such by statements appearing before the current CLASS statement.

The use or absence of "EACH\*" indicates whether or not membership is individual as opposed to collective.

Real, Integer, Logical, Program Object

The statements

REAL  $a_1, a_2, \dots, a_m$

INTEGER  $a_1, a_2, \dots, a_m$

LOGICAL  $a_1, a_2, \dots, a_m$

PROGRAM OBJECT  $a_1, a_2, \dots, a_m$

where element  $a_1$  may have the form  $n_1$  or  $n_1(d_1, d_2, \dots, d_k)$ , defines names  $n_1, n_2, \dots, n_m$  to be of REAL, INTEGER, LOGICAL, or PROGRAM OBJECT mode.

The appearance of dimensions  $(d_1, d_2, \dots, d_k)$  in the element  $a_1$  further defines name  $n_1$  to be an array having  $k$  dimensions.

Dimensions  $(d_1, d_2, \dots, d_k)$  may each assume the following forms:

1. An expression of real or integer mode;
2. An object name; or
3. A class name.

Any name which appears in a dimension is defined by such appearance to be a symbolic dimension unless it is defined elsewhere as an object or class. No other definition of symbolic dimensions is permitted except subsequent use in another dimension.



### Normal Mode

In the absence of explicit mode declarations, names are assigned modes according to their initial letters as required. The correspondence of modes to initial letters is known as the "normal mode".

The statement

$$\begin{aligned} \text{NØRMAL MØDE } m_1(a_1, a_2, \dots, a_i), m_2(b_1, b_2, \dots, b_j), \\ m_3(c_1, c_2, \dots, c_k), \dots, m_r \end{aligned}$$

is used to specify the normal mode. Mode designators  $m$  may be the words REAL, INTEGER, LØGICAL, or PRØGRAM ØBJECT. Alphabetic characters  $a_1, a_2, \dots, b_1, b_2, \dots, c_k$  indicate the initial letters which are to correspond to the various mode designators. Designator  $m_r$  applies to all letters not explicitly mentioned in the statement, and is assumed REAL if absent.

The normal mode so defined will prevail until another NØRMAL MØDE statement is encountered. The initial normal mode for all programs is REAL.

Vector, List

The statements

VECTØR  $n((c_1, c_2, \dots, c_j), d_1, d_2, \dots, d_k)$ , etc.

LIST  $n((c_1, c_2, \dots, c_j), d_1)$ , etc.

define groups of arrays  $c_1, c_2, \dots, c_j$  which have identical dimensions  $d_1, d_2, \dots, d_k$  and are grouped together under the name  $n$ . The name  $n$  is declared to be a vector or list, and names  $c_1, c_2, \dots, c_j$  are declared to be vector components. The number of such name/component/dimension groups which can be declared in one statement is limited only by the maximum statement length.

Unless the mode of name  $n$  is declared explicitly in a REAL, INTEGER, LOGICAL, or PROGRAM OBJECT statement, the normal mode prevailing at the appearance of the VECTØR or LIST statement will be assigned. Components whose modes are not explicitly defined will be assigned the mode of the name  $n$ .

Dimensions  $d_1, d_2, \dots, d_k$  may assume the same form as array dimensions previously described.

Only two differences obtain between vectors and lists:

1. Lists may have only one dimension; vectors may have any number.
2. Lists may be operated on by special processing statements; vectors may not.

Contingent Event, Permanent Event

The statements

CØNTINGENT EVENT  $n((c_1, c_2, \dots, c_j), d_1)$

PERMANENT EVENT  $n((c_1, c_2, \dots, c_j), d_1)$

PERMANENT EVENT  $n$

declare the name  $n$  to be a contingent event with list, permanent event with list, or permanent event. Forms with a list create storage assignments exactly as would a LIST statement.

The event statement is always followed by a series of one or more statements, the last of which must be an END statement. (See CONTROL.) This series of statements embodies the processing associated with the event named  $n$ .

Standard event processing algorithms require the components  $c_1, c_2$ , and  $c_3$  of a CØNTINGENT EVENT list to have modes of REAL, PRØGRAM OBJECT, and PRØGRAM ØBJECT respectively. Any other construction may be used where standard processing is not.

Common

The statement

COMMON  $n_1, n_2, \dots, n_k$

causes storage required by data items named  $n_1, n_2, \dots, n_k$  to be placed in a special area of the computer so that it can be directly accessed by procedures. Additions to this common store are cumulative, items from one common statement being added to those from any previous common statement.

Access to common data by more than one program requires that each program have identical common structure, i.e.:

1. Each item in common must be identically defined in both programs;
2. Common statements in both programs must specify these items in identical order.

Identical definition and order suggest the following rules for common structure:

1. If an item in common has symbolic dimensions, the dimension names should also be in common;

2. If an item in common has dimensions which are object or class names, those names should also be in common;
3. If a PERMANENT EVENT with list is in common, the corresponding item in all programs except that containing the event processing should be declared as a LIST.
4. If a CONTINGENT EVENT with list is in common, the corresponding item in all programs except that containing the event processing should be declared as a LIST and preceded in common by a single variable which is otherwise unused. This extra variable does not appear in the common statements of the program containing the event processing.

Only certain types of names may appear in a common statement, and these types are:

Single variable

Array

Vector

List

Object  
 Class  
 Contingent event with list  
 Permanent event with list  
 Symbolic dimension

The names TIME, ATTACKER, TARGET, or INDEX may not appear in a common statement. Appearance of the name NEXT EVENT in a common statement will place all of the above names in common.

### Procedure

The statement

$$\text{PR}\emptyset\text{CEDURE } n(a_1, a_2, \dots, a_m)$$

designates the entire program in which it appears to be a procedure whose name corresponds to the first six (or less) characters of the name  $n$ . Neither the name  $n$  nor another PR $\emptyset$ CEDURE statement may appear elsewhere in the same program.

The names  $a_1, a_2, \dots, a_m$  and their enclosing parentheses are optional, and designate the dummy arguments to the procedure if present. The following types of names may be used as dummy arguments:

Single variable  
 Array  
 Vector  
 List  
 Object

**Class**

**Contingent Event with list**

**Permanent Event with list**

**Symbolic dimension**

**Statement label.**



## EXPRESSIONS

As many MILITRAN statements depend upon the use of expressions, a brief summary of expression forms and types is presented in this section. A short discussion of retrieval forms is also included.

### Expression Syntax

The overall syntax of expressions is presented here in the familiar Backus type notation.

Brackets  $\langle \rangle$  used below enclose terms designating elements; the sign ::= may be read as "takes the form"; vertical lines may be read as "or"; and all other characters represent themselves.

$$\langle \text{expression} \rangle ::= \langle \text{arithmetic expression} \rangle \mid \langle \text{logical expression} \rangle \\ \langle \text{program object expression} \rangle$$

$$\langle \text{arithmetic expression} \rangle ::= \langle \text{real expression} \rangle \mid \langle \text{integer expression} \rangle$$

$$\langle \text{arithmetic operator} \rangle ::= + \mid - \mid * \mid / \mid .P.$$

$$\langle \text{arithmetic comparator} \rangle ::= .E. \mid .G. \mid .L. \mid .NE. \mid .GE. \mid .LE.$$

$$\langle \text{logical operator} \rangle ::= .\emptyset R. \mid .AND. \mid .EQV. \mid .EX\emptyset R.$$

$\langle \text{real expression} \rangle ::= \langle \text{real data item} \rangle \mid + \langle \text{real expression} \rangle \mid$   
 $- \langle \text{real expression} \rangle \mid ( \langle \text{real expression} \rangle ) \mid$   
 $\langle \text{real expression} \rangle \langle \text{arithmetic operator} \rangle \langle \text{real expression} \rangle \mid$   
 $\langle \text{integer expression} \rangle \langle \text{arithmetic operator} \rangle \langle \text{real expression} \rangle \mid$   
 $\langle \text{real expression} \rangle \langle \text{arithmetic operator} \rangle \langle \text{integer expression} \rangle$

$\langle \text{integer expression} \rangle ::= \langle \text{integer data item} \rangle \mid + \langle \text{integer expression} \rangle \mid$   
 $- \langle \text{integer expression} \rangle \mid ( \langle \text{integer expression} \rangle ) \mid$   
 $\langle \text{integer expression} \rangle \langle \text{arithmetic operator} \rangle \langle \text{integer expression} \rangle$

$\langle \text{logical expression} \rangle ::= \langle \text{logical data item} \rangle \mid .\text{NOT.} \langle \text{logical expression} \rangle \mid$   
 $\langle \text{logical expression} \rangle \langle \text{logical operator} \rangle \langle \text{logical expression} \rangle \mid$   
 $\langle \text{arithmetic expression} \rangle \langle \text{arithmetic comparator} \rangle \langle \text{arithmetic expression} \rangle \mid$   
 $\langle \text{program object expression} \rangle .\text{IN.} \langle \text{object or class name} \rangle \mid$   
 $\langle \text{program object expression} \rangle .\text{IS.} \langle \text{program object expression} \rangle \mid$   
 $( \langle \text{logical expression} \rangle )$

$\langle \text{program object expression} \rangle ::= \langle \text{program object data item} \rangle \mid$   
 $\langle \text{object or class name} \rangle ( \langle \text{arithmetic expression} \rangle ) \mid$   
 $( \langle \text{program object expression} \rangle )$

<data item> ::= <single variable name> |  
                   <symbolic dimension name> |  
                   <subscripted array name> |  
                   <subscripted vector name> |  
                   <function> | <constant>

<subscripted array name> ::= <array-type name> ( <subscript list> )

<subscripted vector name> ::= <vector-type name> ( <subscript list> , <arithmetic expression> )

<array-type name> ::= <array name> | <vector component name>

<vector-type name> ::= <vector name> | <list name> |  
                           <contingent-event-with-list name> |  
                           <permanent-event-with-list name>

<subscript list> ::= <subscript> | <subscript list> , <subscript>

<subscript> ::= <arithmetic expression> | <program object expression>

<function> ::= <external procedure name> ( <argument list> ) |  
                   <open procedure name> ( <argument list> ) |  
                   <external procedure name> | <open procedure name>

$$\langle \text{argument list} \rangle ::= \langle \text{argument} \rangle \mid \langle \text{argument list} \rangle , \langle \text{argument} \rangle$$

$$\langle \text{argument} \rangle ::= \begin{array}{l} \langle \text{expression} \rangle \mid \langle \text{vector-type name} \rangle \\ \langle \text{array-type name} \rangle \mid \langle \text{object or class name} \rangle \mid \\ \langle \text{statement label name} \rangle \end{array}$$

### Functions

A function is a procedure whose execution is implied by its use in an expression. Execution of the function always returns a single value which replaces the function in the expression.

Arguments to a function must correspond in type and order to those expected by the procedure.

A name whose type is not otherwise declared is implicitly declared to be an external procedure (function) when it appears with an argument list in an expression.

### Standard Functions

Several functions are pre-defined in MILITRAN, and reference to their names automatically produces either open coding or calling sequences to library subroutines. These functions are described below. Values are returned in the same mode as the arguments except where noted. Arguments must be REAL except where noted.

ABS (v) returns  $|v|$ . The argument v may be either REAL or INTEGER.

ATAN ( $v_1, v_2$ ) returns the angle  $\alpha$  whose tangent is  $v_1/v_2$ . ( $0 \leq \alpha < 2\pi$  ).

COS (v) returns  $\cos v$ .

EPSILON (v) returns  $(v + \epsilon)$  where  $\epsilon$  is the smallest increment physically recognizable in v. Argument v may be either REAL or INTEGER. ( $\epsilon \equiv 1$  when v is integer.)

EXP (v) returns  $e^v$  where e is the Naperian base.

INTEGER ( $v$ ) returns the largest integer  $i$  such that  $|i| \leq |v|$ . Argument  $v$  may be either REAL or INTEGER. Result  $i$  is returned in INTEGER mode.

LOG ( $v$ ) returns the natural logarithm of  $v$ .

MAX ( $v_1, v_2, \dots, v_j$ ) returns the maximum value among the arguments ( $v_1, \dots, v_j$ ). Arguments may be REAL or INTEGER.

MIN ( $v_1, v_2, \dots, v_j$ ) returns the minimum value among the arguments ( $v_1, \dots, v_j$ ). Arguments may be REAL or INTEGER.

RANDOM returns a REAL pseudo-random value,  $v$ , ( $0 \leq v < 1$ ).

REAL (v) returns the value of v in REAL mode. The argument v may be either REAL or INTEGER. ( $|v| < 2^{27}$ .)

SIGN (v<sub>1</sub>,v<sub>2</sub>) returns the value

$$|v_1| \cdot \frac{|v_2|}{v_2} \text{ if } v_2 \neq 0, \quad |v_1| \text{ if } v_2 = 0.$$

SIN (v) returns sin v.

SQRT (v) returns  $\sqrt{|v|}$

TAN (v) returns tan v.

(Values exceeding the maximum possible REAL value are truncated to that maximum.)



Subscripts

Retrieval of specific data items from arrays and vectors is accomplished by means of subscripts. Types of names requiring subscripts fall into two groups as follows:

<u>Array-type</u>	<u>Vector-type</u>
Array	Vector
Vector component	List
	Contingent event with list
	Permanent event with list

Array-type names require exactly as many subscripts as they have dimensions. Consider the statement

$$\text{REAL } n(d_1, \dots, d_1, \dots, d_m)$$

which defines the name  $n$  to be an  $m$ -dimensional array. Retrieval of a single member of  $n$  would be accomplished by the expression

$$n(e_1, \dots, e_1, \dots, e_m)$$

occurring elsewhere in the program. The expressions  $e_1, \dots, e_m$  are subject to the following rules:

1. Any expression  $e_1$  may be an arithmetic expression.
2. Expression  $e_1$  may be a program object expression if and only if dimension  $d_1$  is an object or class name.

Vector-type names require one more subscript than they have dimensions. Consider the statement

$$\text{VECTOR } n((c_1, \dots, c_j, \dots, c_k), d_1, \dots, d_1, \dots, d_m)$$

which defines the name to be an  $m$ -dimensional vector having  $k$  components. Retrieval of a single member of  $n$  would be accomplished by either of the expressions

$$n(e_1, \dots, e_1, \dots, e_m, e_{m+1})$$

$$c_j(e_1, \dots, e_1, \dots, e_m)$$

occurring elsewhere in the program. The expressions  $e_1, \dots, e_m$  are subject to the same rules as are subscripts for array-type names. The expression  $e_{m+1}$  must be arithmetic and equal to  $j$ .

PROCESSING

Processing statements are summarized in this section. They include:

PLACE	REPLACE
PLACE ENTRY	REPLACE ENTRY
REMOØVE	RESET LENGTH
REMOØVE ENTRY	Substitution

Substitution

All substitution statements take the form

$$a = b$$

where a is a subscripted or unsubscripted name and b is an expressions. The name "a" may not be an object, class, permanent event without list, statement label, procedure, external procedure, or open procedure.

The following processing may be accomplished through a substitution statement:

1. If a and b are of the same mode, the value of expression b replaces the value of a.
2. If a and b are both subscripted vector-type names, the contents of b replace the contents of a without regard to mode. ("Contents" as used above refers only to a single value, not the entire array.)
3. If a is real and b is integer, the value of expression b is converted to a real number

and replaces the value of a.

4. If a is integer and b is real, the value of expression b is truncated to an integer and replaces the value of b.

Conditions 3 and 4 above apply only when condition 2 does not.

### List Processing

Vector-type names defined in LIST, CONTINGENT EVENT, and PERMANENT EVENT statements represent groups of values which may be processed by means of special "list processing statements."

In the discussion of list processing statements which follows, all descriptions will refer to the generalized lists defined by

$$\text{LIST } m((m_1, \dots, m_1, \dots, m_j), d_m), n((n_1, \dots, n_1, \dots, n_k) d_n)$$

The symbols designating the lists and components defined above will be maintained throughout the discussion.

### List Entries

The list m may contain  $d_m$  entries. The  $e^{\text{th}}$  entry in list m is the set of values.

$$m_1(e), m_2(e), \dots, m_j(e)$$

The current number of entries in list m is represented by the function

$$\text{LENGTH}(m)$$

which is initially equal to zero for all lists.

Any value in the group represented by the name  $m$  may be altered by means of a substitution statement. However, substitution statements do not maintain the LENGTH function and list processing statements consider the  $e^{\text{th}}$  entry valid only if  $1 \leq e \leq \text{LENGTH}(m)$ .

### List Processing Conditions

Several statements and functions involved in list processing depend upon logical conditions of the form

$$(b_1, b_2, \dots, b_k, b_x)$$

where  $k$  is the number of components in the list.

The  $e^{\text{th}}$  entry of list  $n$  is said to meet the condition  $(b_1, \dots, b_k, b_x)$  if and only if

1.  $b_1.\text{EQV}.\text{TRUE}$  for  $1 \leq i \leq k$ ; and
2.  $b_x.\text{EQV}.\text{TRUE}$

The logical expression  $b_1$  may involve the current value of  $n_1(e)$ , which value is represented in  $b_1$  by an asterisk. The logical expression  $b_x$  may involve  $e$ , which value is represented in  $b_x$  by an asterisk. Since more than one entry in list  $n$  may meet the condition  $(b_1, \dots, b_k, b_x)$ , the entry number "e" is never used explicitly.

The following abbreviations are permitted in the construction of list processing conditions:

1. Omitted expressions are assumed to be TRUE; e.g., the conditions (TRUE,TRUE) and (,) are

equivalent.

2. Commas separating identically true expressions at the end of the condition may be omitted; e.g., the conditions  $(b_1, b_2, \text{TRUE}, \text{TRUE})$  and  $(b_1, b_2)$  are equivalent.
3. The expression " $*.E.a$ ," where  $a$  is an arithmetic expression, may be represented by " $a$ ."
4. The expression " $*.IS.a$ ", where  $a$  is a program object expression, may be represented by " $a$ ."
5. The expression " $IN.a$ ," where  $a$  is an object or class name, may be represented by " $a$ ."

A condition may be restricted by the use of the functions GST and LST. One and only one expression in a condition may be subjected to a GST/LST restriction. Every condition containing GST or LST is met by no more than one entry in list  $n$ .

1.  $(b_1, \dots, \text{GST}(b_1), \dots, b_k, b_x)$  refers to that entry whose  $n_1$  is the greatest  $n_1$  among all entries meeting condition  $(b_1, \dots, b_1, \dots, b_k, b_x)$ .
2.  $(b_1, \dots, \text{LST}(b_1), \dots, b_k, b_x)$  refers to that entry whose  $n_1$  is the least  $n_1$  among all



entries meeting condition

$(b_1, \dots, b_1, \dots, b_k, b_x)$ .

3.  $(b_1, \dots, b_k, \text{GST}(b_x))$  refers to the highest numbered entry meeting condition  $(b_1, \dots, b_k, b_x)$ .
4.  $(b_1, \dots, b_k, \text{LST}(b_x))$  refers to the lowest numbered entry meeting condition  $(b_1, \dots, b_k, b_x)$ .

Where expression  $b_1$  is subjected to a GST/LST condition, component  $n_1$  must be of real or integer mode. Where duplicate minima or maxima occur, the lowest numbered entry is chosen.

#### Place, Place Entry

Execution of the statement

$\text{PLACE}(c_1, \dots, c_1, \dots, c_p)$  IN  $n$

causes the current value of  $\text{LENGTH}(n)$  to be increased by one and the values  $c_1, \dots, c_j$  to replace current values of  $n_1(\text{LENGTH}(n)), \dots, n_p(\text{LENGTH}(n))$ . The number of expressions ( $p$ ) may not exceed the number of components ( $k$ ) in list  $n$ . Expression  $c_1$  must be of the same mode as component  $n_1$  for  $1 \leq i \leq p$ .



Execution of the statement

PLACE ENTRY  $m(e)$  IN  $n$

is identical in a processing sense to execution of

PLACE( $m_1(e), \dots, m_p(e)$ ) IN  $n$

where  $p$  is equal to the smallest number of components contained in either list ( $m$  or  $n$ ).

#### Remove, Remove Entry

Execution of the statement

REMOVE ( $b_1, \dots, b_k, b_x$ ) FROM  $n$

will cause all entries meeting condition ( $b_1, \dots, b_k, b_x$ ) to be removed from list  $n$ .

Execution of the statement

REMOVE ENTRY  $n(e)$

will cause the  $e^{\text{th}}$  entry in list  $n$  to be removed.

For every entry removed from list  $n$ , the value of the function LENGTH( $n$ ) is reduced by 1. Rearrangement of the list to eliminate blank entries is performed where necessary.

Replace, Replace Entry

Execution of the statement

REPLACE ENTRY  $n(e)$  BY  $(c_1, \dots, c_p)$

causes the values of  $n_1(e), \dots, n_p(e)$  to be replaced by the values of expressions  $c_1, \dots, c_p$ . The number of expressions ( $p$ ) may not exceed the number of components ( $k$ ) in list  $n$ . Modes of expression  $c_i$  and component  $n_i$  must match for  $1 \leq i \leq p$ . The value of  $n_i(e)$  before replacement may be represented in expression  $c_i$  by an asterisk.

Execution of the statement

REPLACE ENTRY  $n(e_1)$  BY ENTRY  $m(e_2)$

causes values  $m_1(e_2), \dots, m_p(e_2)$  to replace the current values  $n_1(e_1), \dots, n_p(e_1)$ , where  $p$  is the smallest number of components contained in either list ( $m$  or  $n$ ).

The two statements

REPLACE ENTRY  $n(e_1)$  BY ENTRY  $(e_2)$

REPLACE ENTRY  $n(e_1)$  BY ENTRY  $n(e_2)$

are identical in a processing sense.

### Execution of the statements

REPLACE( $b_1, \dots, b_k, b_x$ ) BY ( $c_1, \dots, c_p$ ) IN n

REPLACE( $b_1, \dots, b_k, b_x$ ) BY ENTRY m(e) IN n

REPLACE( $b_1, \dots, b_k, b_x$ ) BY ENTRY (e) IN n

cause replacement of every entry in list n which meets condition ( $b_1, \dots, b_k, b_x$ ). Replacement is accomplished in exactly the same manner as by corresponding REPLACE ENTRY statements.

### Reset Length

#### Execution of the statement

RESET LENGTH (n) TO e

will arbitrarily reset the value of function LENGTH(n) to the positive integer value of arithmetic expression e.

Use of this statement is required only when non-list-processing statements have been used to enter values in list n or when the programmer wishes to ignore entries beyond the  $e^{\text{th}}$  entry.

### List Entry Locating Functions

Two MILITRAN functions operate directly upon list processing conditions. These functions are used within the context of an expression as are the functions discussed under EXPRESSIONS.

The functions

MINIMUM INDEX ( $n(b_1, \dots, b_k, b_x), s$ )

RANDOM INDEX ( $n(b_1, \dots, b_k, b_x), s$ )

return an integer value designating an entry in the list  $n$  which meets the condition  $(b_1, \dots, b_k, b_x)$ . If no such entry exists, control transfers immediately to the statement labelled  $s$ .

The distinction between MINIMUM INDEX and RANDOM INDEX obtains only when more than one entry in list  $n$  satisfies condition  $(b_1, \dots, b_k, b_x)$ . MINIMUM INDEX chooses the lowest numbered entry meeting the condition; RANDOM INDEX chooses one entry at random from all those meeting the condition.

MINIMUM INDEX may be shortened to INDEX without loss of meaning.

INPUT/OUTPUT

Input/output statements are summarized in this section. They include:

BACKSPACE	READ
BACKSPACE FILE	READWRITE
BINARY READ	REWIND
BINARY WRITE	UNLØAD
END FILE	WRITE
FØRMAT	

Logical Unit Designations

Input/output units are designated in MILITRAN source programs as follows:

Tape Units by positive integers;

Line printer by the name PRINTER;

Card reader and punch by the name CARDS.

Tape Control Statements

Statements whose execution causes tape units to perform operations not involving transfer of data are tabulated below. In all cases, the designation *t* is an arithmetic expression.

<u>Statement</u>	<u>Effect</u>
BACKSPACE (t)	Designated tape unit backspaces one record.*
BACKSPACE (t)	Designated tape unit backspaces until an end-of-file mark is passed.*
END FILE (t)	An end-of-file mark is written on the designated tape.
REWIND (t)	Designated tape unit rewinds.*
UNLOAD (t)	Designated tape unit rewinds and becomes inoperative.

\* Statements marked with an asterisk have no effect  
if designated tape unit is fully rewound.

## Input/Output Lists

Input/output statements which involve transfer of data between the computer and external devices require a list of those items which are to be transferred. All such lists are identically constructed. The summary below utilizes the notation previously used to describe expressions.

$$\langle \text{I/O list} \rangle ::= \langle \text{expression} \rangle \mid ( \langle \text{I/O list} \rangle ) \mid \\ \langle \text{I/O list} \rangle , \langle \text{I/O list} \rangle \mid \\ (( \langle \text{I/O list} \rangle ) \langle \text{implied DO loop} \rangle ) \mid \\ \langle \text{I/O list} \rangle , \langle \text{void} \rangle$$

$$\langle \text{implied DO loop} \rangle ::= \text{FOR } \langle \text{program object single variable name} \rangle .\text{IN. } \langle \text{object or class name} \rangle \mid \\ \langle \text{terminating condition} \rangle \mid \\ \langle \text{terminating condition} \rangle , \langle \text{index} \rangle \mid \\ \langle \text{terminating condition} \rangle , \langle \text{index} \rangle = \langle \text{expression} \rangle \mid \\ \langle \text{terminating condition} \rangle , \langle \text{index} \rangle = \langle \text{expression} \rangle , \langle \text{expression} \rangle$$

$$\langle \text{terminating condition} \rangle ::= \text{UNTIL } \langle \text{logical expression} \rangle$$

$$\langle \text{index} \rangle ::= \langle \text{any expression permitted on the left side of a substitution statement} \rangle$$



## Binary Read, Binary Write

The statements

BINARY READ (t) data

BINARY WRITE (t) data

where the "data" is any input/output list, cause reading or writing in binary form on magnetic tape. The expression t must designate a tape unit.

Binary reading and writing are performed without conversion, i.e., items are handled in their exact internal form. Each BINARY WRITE statement writes a logical block of data on tape whose length is dependent upon the number of items in the input/output list. A BINARY READ statement may read only one logical block. If fewer items are read than are contained in the block, the remaining items in the block are skipped.

## Read, Write

The statements

READ (t,s) data

WRITE (t,s) data

cause reading or writing of information on the input/output unit designated by the expression t according to a format specified in a FORMAT statement which has the label s. Data must be specified by an input/output list.

The statement

READWRITE(t<sub>1</sub>,s<sub>1</sub>,t<sub>2</sub>,s<sub>2</sub>) data

is identical in a processing sense to the statements

READ (t<sub>1</sub>,s<sub>1</sub>) data

WRITE (t<sub>2</sub>,s<sub>2</sub>) data

executed in the order shown.

## Format

Formats for data transferred by READ, WRITE, AND READWRITE statements are specified by statements of the form

s FØRMAT (Specification)

The label s is required, since it is the only link between the READ or WRITE and its associated FØRMAT.

The specification portion of a FØRMAT statement consists of a series of fields and punctuation marks which indicate the form and placement of data in external records.

Fields are of two types: data and non-data. Data fields specify transmission of data to or from items in the input/output list. Non-data items involve transfer only between the FØRMAT statement and external records. Data fields must be separated from succeeding fields by commas, while non-data fields need not be.

Field groups may be repeated through the use of parentheses. The notation "n(sub-specification)" will cause the group in parentheses to be repeated n times. If n is absent, the group is repeated indefinitely. The sub-specification may not contain parentheses.

The end of any parentheses without a specific number of repetitions (n) normally signifies the end of a record. The specification for the next record starts from the corres-

ponding open parentheses. Additional changes of record may be specified by a slash (/). Two consecutive slashes indicate a blank record.

During execution of a READ, WRITE, or READWRITE statement, input/output lists and FØRMAT specifications are simultaneously scanned from left to right. Each data item in the input/output list corresponds to a data field in the FØRMAT specification, correspondence being established solely by order of occurrence. Transmission ends when the input/output list is satisfied.

Voids in the input/output list cause the corresponding data fields to be skipped (input) or filled with blanks (output). An input/output list which consists solely of an implied "DO-loop" will cause tape motion only if at least one item of data is transferred. If an input/output list results in reading of a partial record, the remainder of the record is skipped.

Non-data fields are designated by the letters X or H. The specification WX causes w characters to be ignored on input or assumed blank on output. The specification WH must be followed immediately by w characters which will be copied literally from FØRMAT statement to record (output) or vice-versa (input).

Data fields are designated by specifications of the form  $ncw.d$  where  $n$  is the number of fields,  $c$  is an identifier designating field type,  $w$  is the field width, and  $d$  is a supplementary width. The supplementary width is not required for some fields. The number of fields is assumed to be 1 if absent. Basic field types are summarized below. Source and target addresses referred to in the table are items in the input/output list and are discussed in detail immediately following the table. The number of characters in the external record covered by any field is always equal to the field width.

Field  
SpecificationInterpretation

AW	<p>External field contains alphameric data; internal representation will be BCD code. Input: The rightmost six characters from the field replace data at the target address. If w is less than six, w characters are left justified and filled to six characters with blanks.</p> <p>Output: Six characters from the source address are right justified in the field. Remaining characters are blank. If w is less than six, the leftmost w characters from the source address are used.</p>
IW	<p>External field contains decimal integer; internal representation is integer data. Input: All blanks are considered zero. Output: Leading zeroes are replaced by blanks.</p>
<del>OW</del>	<p>External field contains octal integer; internal representation is integer data. Input: All blanks are considered zero. Output: Leading zeroes are replaced by blanks.</p>
LW	<p>External field contains word beginning with T or F; internal form is logical. Input: T or F in field results in transfer of TRUE or FALSE to target address. Output: TRUE or FALSE at source address causes T or F to be right justified in field. Remainder of field is filled with blanks.</p>



Field  
Specification

Effect

Jw.d

External form is the name of an object and its ordinality; internal form is a program object value.

Input: Field is scanned from left to right until w-d characters are read or a left parenthesis appears. Scanned characters (blanks ignored) are compared with names of all OBJECT types in program. Digits following a left parenthesis, or the rightmost d characters, are assumed to be the ordinality. Program object value constructed and transmitted to target address.

Output: Ordinality of source address value is converted to decimal integer, enclosed in parentheses, and placed in rightmost d+2 characters of field. Object name is right adjusted in leftmost w-d-2 characters. Remainder of field is blank.

Jw

Input: Ordinality is assumed unity if no left parenthesis appears.

Output: Only object name appears in field.

Fw.d

External form is decimal number; internal form is real.

Input: Decimal point is assumed d characters from the right unless present.

Output: Decimal point is inserted as d<sup>th</sup> character from the right.

Ew.d

External form is decimal number and exponent. Internal form is real.

Input: Number is assumed to have the form xxxx+xx where sign separates base value and exponent. Base value times ten to exponent value is transmitted in real mode to target address. Exponent is assumed unity if absent. Decimal point in base value is assumed such that d digits are fractional unless decimal point appears explicitly. Exponent may have one or two digits.

Output: Field has the form .xxxxE+xx, where decimal point falls to the left of the d<sup>th</sup> character in the base value.



Data fields of type E,F,I, and  $\emptyset$  may be signed. Missing sign is assumed plus on input; plus sign is not written on output.

Source and target addresses are determined by items in the input/output list. A source address is a value to be written; a target address is a position into which a value is to be read.

All source addresses yield either the value of an expression or a void. Fields corresponding to voids in the input/output list will be blank.

A target address is implicitly void if the expression corresponding to it contains any operator, external procedure, or open procedure outside of its subscripts. Fields corresponding to voids are ignored.

CONTROL

Statements whose major function is the control of program operating sequence are summarized in this section.

They include:

CØNTINUE	IF
DØ	NEXT EVENT
END	NEXT EVENT EXCEPT
END CØNTINGENT EVENTS	PAUSE
END FILE RETURN	RETURN
END RECØRD RETURN	STØP
EXECUTE	UNLESS
GØ TØ	

Go To

Execution of the statement

GØ TØ s

causes the program to continue from the statement whose label is s. In the discussions which follow, this operation will be described as: "Control is transferred to s."

If, Unless

Execution of either of the statements

IF(b),x,y

UNLESS(b),y,x

will transfer control to x if logical expression b has the value TRUE, to y if b is FALSE. The second comma and label may be omitted, in which case the statement immediately following the IF or UNLESS is assumed.

Pause, Stop

Execution of the statement

PAUSE n

causes the computer to stop with the octal number n displayed. Execution may be restarted by manual means. Number n may not exceed 30000 octal.

Execution of the statement

STØP

causes execution of the program to be terminated. Restart cannot be effected.

Execute, Return

The PRØCEDURE statement, described under STRUCTURE, is used to define MILITRAN programs whose operation is to be controlled by other programs. The control statements EXECUTE and RETURN implement control of such programs.

Execution of the statement

EXECUTE n(a<sub>1</sub>,a<sub>2</sub>,...,a<sub>m</sub>)

will cause control to be transferred to the PROCEDURE whose name corresponds to the first six characters of the name n. Arguments a<sub>1</sub>,a<sub>2</sub>,...,a<sub>m</sub> must correspond in mode, type, and order to the dummy arguments of the procedure. The name n is

declared by its appearance in an EXECUTE statement to be an external procedure name.

In a program which is a procedure, execution of the statement

RETURN

will return control to the program in which the EXECUTE statement appears.

Procedures which are used as functions (see under EXPRESSIONS) must return a value to the executing program.

The statement

RETURN e

where e is an expression accomplishes transfer of both control and the value of e.

Do, Continue

Execution of the statement

$$D\emptyset (s) \text{ UNTIL } b, i = e_1, e_2$$

causes iterative execution of statements following the  $D\emptyset$  up to and including the statement labelled  $s$ . Before the first iteration, index  $i$  will be set to the value of expression  $e_1$ ; before subsequent iterations, index  $i$  will be incremented by the value of expression  $e_2$ . Index  $i$  may be any unsubscripted or subscripted name of type single variable, array, vector, list, or event; with list.

If at the beginning of any iteration the value of logical expression  $b$  is TRUE, control transfers immediately to the statement following  $s$ .

Execution of the statement

$$D\emptyset (s) \text{ F\emptyset R } a.IN.b$$

causes iterative execution of statements following the  $D\emptyset$  up to and including the statement labelled  $s$ . The single variable  $a$  must be of program object mode and will successively assume the identity of all members of the object or class  $b$ .

When all members of  $b$  have been represented by  $a$ , control transfers immediately to the statement following  $s$ .

In both forms of the  $D\emptyset$  statement above it is necessary that the statement labelled  $s$  permit control to pass through it to the next statement. Thus the statements  $G\emptyset T\emptyset$ , NEXT EVENT, and IF or UNLESS with two labels are prohibited as terminal statements of a  $D\emptyset$  loop.

Restrictions on statements terminating  $D\emptyset$  loops do not limit the variety of processing arrangements possible, since the statement

#### C $\emptyset$ NTINUE

can be used at any point in a program. This statement performs no operations and requires no space in the computer. Its label, however, may be used to terminate a  $D\emptyset$  loop.

Other  $D\emptyset$  statements may appear between one  $D\emptyset$  and its terminating statement, but the "inner" loop must terminate at or before the end of the "outer" loop.



End, Next Event, Next Event Except

A group of executable statements beginning with one of the statements

CONTINGENT EVENT  $n((c_1, c_2, \dots, c_j), d)$

PERMANENT EVENT  $n((c_1, c_2, \dots, c_j), d)$

PERMANENT EVENT  $n$

and ending with the statement

END

is known in MILITRAN as an "event." Depending upon the initial statement, the event is either a "contingent event" or a "permanent event."

Events are processed in a sequence determined by the structure of the MILITRAN source program. The "natural" or unmodified sequence is:

1. The first permanent event in the program.
2. Subsequent permanent events in the order of their appearance in the program.
3. The last permanent event in the program.
4. The "next contingent event".

This sequence is repeated until terminated by either failure to select a "next contingent event" or transfer of control to a portion of the program not in any event.

It is not required that a program have any minimum number of permanent or contingent events. In the discussion which follows, we will assume that irrelevant items in the natural sequence are ignored.

Selection of the "next contingent event" is dependent upon the current value of TIME. Of all entries in all contingent event lists, one is selected whose first component exceeds TIME by the smallest positive value. The first component is assumed to be of real mode; duplicate minima within one event cause the entry of least index to be chosen; duplicate minima in more than one list cause an entry to be chosen from the event which appears earliest in the natural sequence.

Execution of the statement

NEXT EVENT

causes control to be transferred to the next event in the natural sequence. If the NEXT EVENT statement is not itself contained in an event, control is passed to the first permanent event.

The statements

NEXT EVENT ( $n_1, n_2, \dots, n_m$ )

NEXT EVENT EXCEPT ( $n_1, n_2, \dots, n_m$ )

behave exactly as does the NEXT EVENT statement, but modify the natural sequence. NEXT EVENT EXCEPT will assume that events  $n_1, n_2, \dots, n_m$  do not exist. NEXT EVENT ( $n_1, n_2, \dots, n_m$ ) will assume that only the events named in parentheses exist and that they occur in the order listed.

When control is transferred to a contingent event by means of an event sequencing statement, the values of TIME, ATTACKER, TARGET, and INDEX are automatically set. Assuming that control has been transferred to CONTINGENT EVENT  $n$  (above) because its  $i^{\text{th}}$  entry contains the minimum first component, then:

TIME =  $c_1(1)$

ATTACKER =  $c_2(1)$

TARGET =  $c_3(1)$

INDEX = 1

Transfer of values above is made without respect to modes, e.g., ATTACKER is not valid unless  $c_2$  is of program object mode.

End Contingent Events, End File Return, End Record Return

Certain conditions occurring during the running of a program are detected as errors by the program. Three of these conditions are:

1. In attempting to choose a "next contingent event," the program finds no entries whose values equal or exceed the current value of TIME.
2. In reading from magnetic tape, the program encounters an end-of-file mark on the tape before the input/output list is satisfied.
3. In executing a BINARY READ, the end of a logical block is encountered before the input/output list is satisfied. (See under BINARY READ)

In all of the above cases, control is normally wrested from the program and execution is terminated. However, execution of the statements

END CØNTINGENT EVENTS (S)

END FILE RETURN (S)

END RECØRD RETURN (S)

causes the program to be modified in such a way as to return control to the statement labelled s if and when the appropriate error condition occurs.

COMPILER

Statements whose function is providing information to the processor are summarized in this section. These essentially machine-dependent statements are:

END CØMPILATIØN  
SUSPEND FAP LISTING

End Compilation

The statement

END CØMPILATIØN

signals the end of a MILITRAN source program. The statement may not contain comments and may not occupy more than one card.

Columns 73-75 of the END CØMPILATIØN card will be preserved and used to identify the translated program.

Suspend FAP Listing

The statement

SUSPEND FAP LISTING

appearing anywhere in a MILITRAN source program will cause listing of the translated program to be omitted by the processor.

APPENDIXEnvironment Declarations

REAL  $n_1(i_1, i_2, \dots, i_k), \dots, n_m(i_1, i_2, \dots, i_j)$

INTEGER  $n_1(i_1, i_2, \dots, i_k), \dots, n_m(i_1, i_2, \dots, i_j)$

LOGICAL  $n_1(i_1, i_2, \dots, i_k), \dots, n_m(i_1, i_2, \dots, i_j)$

OBJECT  $n_1(i_1), n_2(i_2), \dots, n_m(i_m)$

PROGRAM OBJECT  $n_1(i_1, i_2, \dots, i_k), \dots, n_m(i_1, i_2, \dots, i_j)$

CLASS (c) CONTAINS  $a_1, a_2, \dots, a_m$

NORMAL MODE  $m_1(a_1, a_2, \dots, a_k), m_2(b_1, b_2, \dots, b_r)$

VECTOR  $n((a_1, a_2, \dots, a_1), d_1, d_2, \dots, d_1)$

COMMON  $n_1, n_2, \dots, n_1$

Arithmetic

A = B

Logical

A = B

Control Statements

GO TO s

PAUSE j



STOP  
 IF (b)  $s_t, s_f$   
 UNLESS (b)  $s_f, s_t$   
 DO (s) UNTIL b,  $n = e_1, e_2$   
 DO (s) FOR a.IN.b  
 CONTINUE

### List Processing Statements

LIST  $n((c_1, c_2, \dots, c_1), d)$   
 LENGTH (n)  
 RESET LENGTH (n) to p  
 PLACE  $(e_1, e_2, \dots, e_1)$  IN n  
 REMOVE ENTRY  $n(k)$   
 PLACE ENTRY  $m(j)$  IN n  
 REPLACE ENTRY  $n(k)$  BY  $(e_1, e_2, \dots, e_1)$   
 REPLACE ENTRY  $n(k)$  BY ENTRY  $m(j)$   
 REMOVE  $(b_1, b_2, \dots, b_1)$  FROM n  
 REPLACE  $(b_1, b_2, \dots, b_1)$  BY  $(e_1, e_2, \dots, e_1)$  IN n  
 REPLACE  $(b_1, b_2, \dots, b_1)$  BY ENTRY  $m(j)$  IN n  
 MINIMUM INDEX  $(n(b_1, b_2, \dots, b_1), s)$   
 RANDOM INDEX  $(n(b_1, b_2, \dots, b_1), s)$   
 GST  
 LST

Event Statements

PERMANENT EVENT  $n((a_1, a_2, \dots, a_i), d)$   
CONTINGENT EVENT  $n((a_1, a_2, \dots, a_i), d)$   
NEXT EVENT  
NEXT EVENT  $(n_1, n_2, \dots, n_i)$   
NEXT EVENT EXCEPT  $(n_1, n_2, \dots, n_i)$   
END  
END CONTINGENT EVENTS (s)

Procedure Statements

PROCEDURE n  
PROCEDURE  $n(a_1, a_2, \dots, a_n)$   
EXECUTE n  
EXECUTE n  $(a_1, a_2, \dots, a_n)$   
RETURN  
RETURN a

Input-Output Statements

FORMAT (Format Specification)  
READ (t,s) List  
WRITE (t,s) List  
READWRITE  $(t_1, s_1, t_2, s_2)$  List  
BINARY READ (t) List

BINARY WRITE (t) List

END FILE RETURN (s)

END RECORD RETURN (s)

BACKSPACE (t)

BACKSPACE FILE (t)

END FILE (t)

REWIND (t)

UNLOAD (t)

Standard Functions

ABS(v)

ATAN(v<sub>1</sub>,v<sub>2</sub>)

CØS(v)

EPSILØN(v)

EXP(v)

INTEGER(v)

LØG(v)

MAX(v<sub>1</sub>,v<sub>2</sub>,...,v<sub>j</sub>)MIN(v<sub>1</sub>,v<sub>2</sub>,...,v<sub>j</sub>)MØD(v<sub>1</sub>,v<sub>2</sub>)

RANDØM

REAL(v)

SIGN(v)

SIN(v)

SQRT(v)

TAN(v)

INDEX

"A" Fields in FØRMAT 48  
ABS 26, 4, 67  
addition (+) 7, 22  
.AND. 8, 22  
arrays 5, 14  
    as procedure arguments 20  
    in CØMMØN statements 19  
    in expressions 24  
    subscripts of 29  
asterisk (\*) 7  
    as multiplication symbol 22  
    in list processing conditions 33-34  
ATAN 26, 4, 67  
ATTACKER 4  
    automatic updating 59  
    in CØMMØN statements 20  
  
BACKSPACE 41, 10, 40, 66  
BACKSPACE FILE 41, 10, 40, 66  
BINARY READ 43, 10, 40, 65  
BINARY WRITE 43, 10, 40, 66  
blanks 3, 6  
BY 9, 37-38, 64  
BY ENTRY 9, 37-38, 64  
  
card punch 40  
card reader 40  
CARDS 40, 4  
characters, alphabetic 3, 4, 9  
characters, alphameric 3, 4, 6, 46, 48  
characters, numeric 3, 6

characters, set of 3  
characters, special 3, 7, 6, 46, 48  
CLASS 13, 10, 12, 63  
classes 5  
    as class members 13  
    as dimensions 14, 19  
    as DØ-loop parameters 55  
    as procedure arguments 21  
    defining statement 13  
    in CØMMØN statements 19, 20  
    in expressions 23, 25  
    in substitution statements 31  
comma (,) 7  
CØMMØN 18-20, 10, 12, 63  
comparators 7, 22, 23  
constants 6, 3, 24  
CØNTAINS 9, 13, 63  
CØNTINGENT EVENT 17, 57, 10, 12, 65  
contingent events 5  
    as procedure arguments 21  
    automatic processing of 57-59  
    defining statements 17, 57  
    in CØMMØN statements 19, 20  
    in expressions 24  
    in substitution statements 31  
    subscripts of 29  
CØNTINUE 56, 10, 51, 64  
CØS 26, 4, 57  
  
dimensions 14  
    in CØMMØN statements 18, 19  
    of arrays 14  
    of lists 16

of objects 12  
 of vector components 16  
 of vectors 16  
 related to subscripts 29-30  
 division (/) 7, 22  
 DØ 55-56, 10, 51, 64  
  
 .E. 7, 22  
     implied in list processing conditions 34  
 "E" fields in FØRMAT 49  
 EACH 13, 4  
 ellipsis (...) 7  
 END 17, 57, 10, 51, 65  
 END CØMPILATIØN 62, 4, 10  
 END CØNTINGENT EVENTS 60-61, 10, 51, 65  
 END FILE 41, 10, 40, 66  
 END FILE RETURN 60-61, 10, 51, 66  
 END RECØRD RETURN 60-61, 10, 51, 66  
 EPSILØN 26, 4, 67  
 .EQV. 8, 22  
 EXECUTE 53-54, 10, 51, 65  
 .EXOR. 8, 22  
 EXP 26, 4, 67  
 exponentiation (.P.) 7, 22  
 expressions 22-25  
     as dimensions 12, 14  
     as DØ-loop parameters 55  
     as logical unit designators 40  
     as subscripts 29-30  
     in input/output lists 42  
     in RETURN statements 54  
     in substitution statements 9, 31-32  
 external procedures 5, 24, 31, (see also "procedures")



"F" fields in FØRMAT 49

FALSE 4, 6

FØR 9, 42, 55, 64

FØRMAT 45-50, 44, 4, 10, 40, 65

FRØM 9, 36, 65

functions 25-28 (see also "procedures")

- for locating list entries 39
- in expressions 24
- standard in MILITRAN 26-28
- use of RETURN statement 54

.G. 7, 22

.GE. 7, 22

GØ TØ 51, 10, 63

- restriction in DØ-loops 56

GST 34-35, 4, 64

"H" fields in FØRMAT 46

"I" fields in FØRMAT 48

IF 51-52, 10, 64

- restriction in DØ-loops 56

implied DØ-loops in input/output lists 42

IN 9, 35, 36, 38, 64

.IN. 7, 22, 23

- implied in list processing conditions 34

INDEX 4

- as abbreviation for MINIMUM INDEX 39
- automatic updating 59
- in CØMMØN statements 20

input/output lists 42-46, 50

INTEGER 14, 27, 4, 10, 12, 16, 67

.IS. 7, 22, 23

- implied in list processing conditions 34

"J" fields in FØRMAT 49

.L. 7, 22

"L" fields in FØRMAT 48

.LE. 7, 22

LENGTH 32-33, 4, 35, 36, 38, 64

line printer 40

LIST 16, 10, 12, 64

list entries 32

- as basis for NEXT EVENT selection 58-59
- conditions specifying 33
- functions for locating 39

list processing 32-39, 64

lists 5

- as procedure arguments 20
- associated with events 17, 58-59
- defining statements 16, 17
- in CØMMØN statements 19
- in expressions 24
- in substitution statements 31
- processing of 32-39
- subscripts of 29

LØG 27, 4, 67

LOGICAL 14, 10, 12, 16, 63

logical block 43, 60-61

logical unit designations 40, 43, 44

LST 34-35, 4, 64

main program 2

MAX 27, 4, 67

MIN 27, 4, 67

MINIMUM INDEX 39, 4, 64

minus (-) 7, 23

mnemonic delimiters 9, 3

MØD 4, 67  
modes 4-5  
    declaration of 14-16  
multiplication (\*) 7, 22  
names 3-4  
.NE. 7, 22  
NEXT EVENT 57-59, 4, 10, 51, 65  
    in CØMMØN statements 20  
    restriction in DØ-loops 56  
NEXT EVENT EXCEPT 57-59, 10, 51, 65  
NØRMAL MØDE 15, 11, 12, 16  
.NØT. 8, 23  
  
"Ø" fields in FØRMAT 48  
OBJECT 12, 11, 63  
objects 5  
    as class members 13  
    as dimensions 14, 19  
    as DØ-loop parameters 61  
    as procedure arguments 21  
    defining statement 12  
    in CØMMØN statements 19, 20  
    in expressions 23  
    in substitution statements 31  
open procedures 5, 24, 31 (see also "procedures")  
operators 3, 7-8, 22-23  
.ØR. 8, 22  
  
.P. 7, 22  
parentheses () 7, 23-24, 45  
PAUSE 53, 11, 51, 63  
period (.) 7, 6  
PERMANENT EVENT 17, 11, 12, 57, 65

permanent events 5  
    as procedure arguments 21  
    automatic processing of 57-59  
    defining statements 17, 57  
    in COMMON statements 19, 20  
    in expressions 24  
    in substitution statements 31  
    subscripts of 29  
PLACE 35, 11, 31, 64  
PLACE ENTRY 36, 11, 31, 64  
plus (+) 7, 23, 51  
PRINTER 4, 40  
PROCEDURE 20, 11, 12, 53, 65  
procedures 2, 5  
    arguments of 20-21, 24, 25, 53  
    as functions 25  
    COMMON statements in 18  
    control statements 53-54  
    defining statements 20-21, 53-54  
    implicit declaration of 25  
    in expressions 24  
    in substitution statements 31  
program 2  
PROGRAM OBJECT 14, 11, 12, 16, 63  
punctuation 3, 7, 22-25  
  
RANDOM 27, 4, 67  
RANDOM INDEX 39, 4, 64  
READ 44, 11, 40, 45-46, 65  
READWRITE 45, 11, 40, 45-46, 65  
REAL 14, 28, 4, 11, 12, 16, 67  
REMOVE 36, 11, 31, 64  
REMOVE ENTRY 36, 11, 31, 64  
REPLACE 33, 11, 31, 64

REPLACE ENTRY 37, 11, 31, 64  
RESET LENGTH 38, 11, 31, 64  
RETURN 53-54, 11, 51, 65  
REWIND 41, 11, 40, 66  
SIGN 28, 4, 67  
SIN 28, 4, 67  
single variables 5  
    as procedure arguments 20  
    in COMMON statements 19  
    in expressions 24  
source program 2  
source address 50, 47-49  
SQRT 28, 4, 67  
statement labels 5  
    as procedure arguments 21  
    defined by CONTINUE statements 56  
    defining range of DO-loops 55  
    in END CONTINGENT EVENTS statements 60-61  
    in END FILE RETURN statements 60-61  
    in END RECORD RETURN statements 60-61  
    in GO TO statements 51  
    in IF statements 51-52  
    in substitution statements 31  
    in UNLESS statements 51-52  
    required for FORMAT statements 45  
statement type identifiers 9-11, 3  
STOP 53, 11, 51, 64  
subscripts 29-30  
substitution statement 31-32, 7, 9, 63  
subtraction (-) 7, 22  
SUSPEND FAP LISTING 62, 11  
symbolic dimensions 5  
    as procedure arguments 21  
    in COMMON statements 18, 20

- in expressions 24
- of arrays, vectors, lists, etc. 14
- of objects 12

TAN 28, 4, 67

tape units 40-41, 43

TARGET 4

- automatic updating 59
- in CØMMØN statements 20

target address 52, 47-49

TIME 4

- as basis for NEXT EVENT selection 58
- automatic updating 59
- in CØMMØN statements 20

TØ 9, 38, 64

TRUE 4, 6

- implied in list processing conditions 33-34

UNLESS 51-52, 11, 64

- restriction in DØ-loops 56

UNLØAD 41, 11, 42

UNTIL 9, 42, 55, 64

VECTØR 16, 11, 12, 63

vector components 5

- defining statements 16
- in expressions 24
- subscripts of 29

vectors 5

- as procedure arguments 20
- defining statement 16
- in CØMMØN statements 19

in expressions 24  
in substitution statements 31  
subscripts of 29  
voids in input/output lists 42, 46, 51

WRITE 44, 11, 40, 45-47, 65

"X" fields in FØRMAT 46



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) Systems Research Group, Inc. 1501 Franklin Avenue Mineola, Long Island, N. Y.	2a. REPORT SECURITY CLASSIFICATION <b>Unclassified</b>
	2b. GROUP

3. REPORT TITLE  
**MILITRAN REFERENCE MANUAL**

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)  
**Technical Report**

5. AUTHOR(S) (Last name, first name, initial)  
**Systems Research Group, Inc.**

6. REPORT DATE <b>June, 1964</b>	7a. TOTAL NO. OF PAGES <b>77</b>	7b. NO. OF REFS
-------------------------------------	-------------------------------------	-----------------

8a. CONTRACT OR GRANT NO. <b>Nonr 2936(00)</b>	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. <b>Navy NR 276-001</b>	
c. <b>AF Proj. 2801,</b>	
d. <b>Task 280101</b>	
9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) <b>USAF Technical Documentary Report No. ESD-TDR-64-390</b>	

10. AVAILABILITY/LIMITATION NOTICES  
**Qualified requesters may obtain copies of this report from the Defense Documentation Center (DDC)**

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY <b>Office of Naval Research, Wash., D.C. &amp; Electronic Systems Division, Air Force Systems Command, Bedford, Mass.</b>
-------------------------	---

13. ABSTRACT

MILITRAN is an algorithmic computer language specifically oriented to the problems encountered in simulation programming. In addition to providing overall flexibility in expressing complex procedures, the language contains features which greatly simplify the maintainence of status lists, handling of numeric and non-numeric data, and sequencing of events in simulated time.

This report is intended as a reference summary for those already familiar with MILITRAN.

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Militran Language Simulation Computers Programming Languages Data Processing Systems Information Retrieval Instruction Manuals Compiler Systems Analysis War Gaming						

**INSTRUCTIONS**

**1. ORIGINATING ACTIVITY:** Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization (*corporate author*) issuing the report.

**2a. REPORT SECURITY CLASSIFICATION:** Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

**2b. GROUP:** Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

**3. REPORT TITLE:** Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

**4. DESCRIPTIVE NOTES:** If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

**5. AUTHOR(S):** Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

**6. REPORT DATE:** Enter the date of the report as day, month, year, or month, year. If more than one date appears on the report, use date of publication.

**7a. TOTAL NUMBER OF PAGES:** The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

**7b. NUMBER OF REFERENCES:** Enter the total number of references cited in the report.

**8a. CONTRACT OR GRANT NUMBER:** If appropriate, enter the applicable number of the contract or grant under which the report was written.

**8b, 8c, & 8d. PROJECT NUMBER:** Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

**9a. ORIGINATOR'S REPORT NUMBER(S):** Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

**9b. OTHER REPORT NUMBER(S):** If the report has been assigned any other report numbers (*either by the originator or by the sponsor*), also enter this number(s).

**10. AVAILABILITY/LIMITATION NOTICES:** Enter any limitations on further dissemination of the report, other than those

imposed by security classification, using standard statements such as:

- (1) "Qualified requesters may obtain copies of this report from DDC."
- (2) "Foreign announcement and dissemination of this report by DDC is not authorized."
- (3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through \_\_\_\_\_."
- (4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through \_\_\_\_\_."
- (5) "All distribution of this report is controlled. Qualified DDC users shall request through \_\_\_\_\_."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

**11. SUPPLEMENTARY NOTES:** Use for additional explanatory notes.

**12. SPONSORING MILITARY ACTIVITY:** Enter the name of the departmental project office or laboratory sponsoring (*paying for*) the research and development. Include address.

**13. ABSTRACT:** Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as (TS), (S), (C), or (U).

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

**14. KEY WORDS:** Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.