MARCH 1974

MATT-1034

AREAD, AN INPUT DATA
PROCESSING ROUTINE

BY

WILLIAM G. PRICE, JR.

PLASMA PHYSICS LABORATORY

MASTER

PRINCETON UNIVERSITY
PRINCETON, NEW JERSEY

This work was supported by U. S. Atomic Energy Commission Contract AT(11-1)-3073. Reproduction, translation, publication, use, and disposal, in whole or in part, by or for the United States Government is permitted.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED.

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.

AREAD, An Input Data Processing Routine

William G. Price, Jr.
Plasma Physics Laboratory, Princeton University
Princeton, New Jersey 08540

ABSTRACT

AREAD is an improved version of the input processing routine FIDO which allows large amounts of data to be specified in a compact and free-form notation. This report covers the use of AREAD as an input module, the data formats acceptable to it, and its internal design.

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or 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.

I. INTRODUCTION

AREAD is an input processing routine which allows large amounts of data to be specified in a compact and free-form notation. It is an improved version of the input subroutine FIDO for the transport codes DTF-II and ANISN* and their descendants. FIDO reads data into variable arrays specified by input "array number" codes; it allows the specification of repeated sequences, multiple zeros, etc. with simple data option codes; it is compatible with a powerful variable dimensioning technique; and its advanced versions perform "free-format" character-by-character scans of the input records. AREAD provides a more flexible linkage between itself and its caller and a greatly expanded vocabulary of option codes. It has been written completely in IBM/360 FORTRAN-IV; machine-dependent features have been isolated to facilitate conversion to other computers.

The first part of this memorandum covers the rules for preparing data to be read by AREAD. The second part tells how to use AREAD as an input module for another program. The final part describes the internal coding of AREAD.

---

*Radiation Shielding Information Center, "ANISN, A One-Dimensional Discrete Ordinates Transport Code," Computer Code Collection CCC-82, Oak Ridge National Laboratory (1972).

II. DATA PREPARATION GUIDE

Arrays

A program which is using AREAD as its input processor will have organized its data requirements into "arrays", each of which is given a particular identifying number. The user will be told the array number, the number of elements, the element type (real, integer, character), and the data order (e.g., column by column).

AREAD scans the input stream looking for control options or array identifiers. The former affect the manner in which the scan is conducted; the latter indicate the destination of the data which follow. When an array identifier (one of the characters \$, *, % or @) is encountered, the origin and length of the array are obtained from the calling program, and AREAD scans for data to be placed in the array. This array is closed out when a new identifier or a scan terminator (a "T" or an end-of-file) is found; an error flag is turned on if the number of elements read was neither the full number nor zero.

It is possible for a main program to provide AREAD with only one array, of indeterminate length. In this case any array identifier in the input stream is assumed to refer to this single array, and no check is made on the array number or on the number of elements filled.

Data Codes

In the following section "A" will denote the active array, which is assumed to have M elements. N is the index of the next element to be filled: it is initialized to 1 when the array identifier is processed.

Every input line scanned by AREAD is broken down into a number of primary "fields", each with up to three "subfields": I, an integer; C, a code character; and V, a real or integer value, as appropriate. Subfields which are not required are left blank. (W stands for the V-subfield in the next field.)

If the code character is "+" or "&", then I, C and V are immediately changed to $V = V * (10.**(+I))$, C = blank, I = 0.

If the code character is "-", the subfields are changed to $V = V * (10.**(-I))$, C = blank, I = 0.

If the code character is a "blank", then I is ignored, $A(N) = V$, and $N = N + 1$ to bring up the next element. The effects of the other codes are described below.

<u>Code</u>	<u>Effect</u>
\$ \$\$ * ** % %% @ @@	new array: first all the processing for the "T" code except the scan termination is performed. Then the dimension M for the new array A specified by I is determined, and N is set to 1. The card scan is resumed in Fixed-Format if the code is a single character or in Free-Format if the code is a double character. If the code is \$, integer values are read into A. If the code is *, floating point values are read into A. If the code is %, A(1) through A(M) are read with the variable format specified with the B option from the unit specified with the D option. If the code is @, A(1) through A(M) are read "unformatted" from the unit specified with the D option.
A	address: $N = V$, I is ignored. Any value V is allowed, but a warning is given if $N \leq 0$ or $N > M$
B	variable format: the following card is read immediately with a (20 A4) format and saved for use as the variable format with the W option or for "%" arrays.
C	count: I and N-1, the number of elements currently filled, are printed.
D	special input unit: if $-100 < I < 100$ and $I \neq 0$, set unit number for special "%" and "@" arrays equal to $ I $, and if $I < 0$, this unit is rewound. (This option can be used artificially to rewind and close units which are no longer needed.)

- E   end array: $N = M + 1$, I and V are ignored
- F   if $N > M$, a warning is printed
      If $N \leq M$, elements $A(N)$ through $A(M)$ are set = V , and
       $N = M + 1$; I is ignored
- G   input unit after EOF: if $0 < I < 100$, the input unit
      number is set to I after an end-of-file; otherwise (the
      initial condition) the scan will terminate after an
      E-O-F.
- H   Hollerith input: the Hollerith mode switch is flipped
      between off and on. When off, the apostrophe (') is
      illegal except in column one, where it signifies a
      comment card to be printed. When on (the initial condi-
      tion) the apostrophe is recognized as the delimiter for
      Hollerith input
- I   linear interpolation ($I \geq 0$ is valid): I points are
      inserted between V and W : $A(N) = V$; then as K goes
      from 1 to I , $A(N + K) = V + (W - V) * K / (I + 1)$;
      finally $N = N + I + 1$
- J   sequence repeat with increment: if either I or V is 0,
      set $V = V + I$ and $I = 1$, then repeat the sequence
       $[A(N - V), \dots, A(N - 1)]$ I times starting at $A(N)$, but
      increment the values in the sequence by one before each
      repeat; finally set $N = N * V$
- K   fixed increment ($I \geq 0$ is valid): as K goes from 1 to I ,
       $A(N + K - 1) = A(N + K - 2) + V$; finally $N = N + I$

- L logarithmic interpolation ($I \geq 0$ is valid): I points are inserted between V and W : $A(N) = V$; then as K goes from 1 to I , $A(N + K) = V * (W/V)^{K/(I + 1)}$; finally $N = N + I + 1$
- M reversed-sign inverted sequence repeat; if either I or V is 0, set $V = I + V$ and $I = 1$, then repeat the V elements of the sequence $[-A(N-1), \dots, -A(N-V)]$ I times starting at $A(N)$, and set $N = N + I * V$
- N inverted sequence repeat: if either I or V is 0, set $V = I + V$ and $I = 1$ then repeat the V elements of the sequence $[A(N - 1), \dots, A(N - V)]$ I times starting at $A(N)$, and set $N = N + I * V$
- O set the print trigger: if $I > 0$ it is set on; if $I < 0$ it is set off; if $I = 0$ it is flipped. If the trigger is on, input cards are listed; it is initialized off
- P scale factor: for $K = 1$ to $N - 1$, $A(N) = V * A(N)$.
- Q sequence repeat: if either I or V is 0, set $V = V + 1$ and $I = 1$, then repeat the V elements of the sequence $[A(N - V), \dots, A(N - 1)]$ I times starting at $A(N)$; finally set $N = N + I * V$
- R repeat ($I > 0$ is valid): V is repeated I times starting at $A(N)$, then $N = N + I$
- S skip: $N = N + I$; V is ignored
- T terminate: write out $N - 1$, the number of elements filled in the current array. If $N \neq 1$ and $N \neq M + 1$, an error flag is turned on; if the write flag or the error flag is on, the elements are printed out. I is printed and the data scan terminates

- W write flag: I is printed out; V is ignored. The write flag, which is turned off whenever a new array is started, is turned on. When the current array processing finishes, its elements will be written out in a standard format if $I \geq 0$, or with the variable format if $I < 0$.
- X change input unit: the input logical unit number is changed to I. V is ignored. (Only the last X or Y on a line will be effective.)
- Y change input unit and rewind: the input logical unit number is changed to I, and a REWIND I is executed. V is ignored. (Only the last X or Y on a line will be effective.)
- Z zero: set $A(N)$ through $A(N + I - 1) = 0$ and set $N = N + I$. V is ignored.
- ' when the Hollerith switch is off, an apostrophe may only appear in column one to indicate a comment card. When the switch is on, a pair of apostrophes delimit a string of characters to be read into $A(N)$, $A(N + 1)$, ... as if with an (A1) format. (An interior double-prime is used to actually read an apostrophe.)
- ? entering a question mark will elicit the date of the version of this data processing routine which is actually being used.
- " the columns past a quotation mark are ignored and may be used for comments. If the " is in column one, the whole card will be printed amidst the AREAD messages.

Formats

The initial input unit for AREAD is set by the main program; it can be changed by the X and Y option codes. Normally AREAD assumes that its input lines are 80-character card images, and it ignores the last 8. Characters are scanned in Fixed-Format or Free-Format, depending on whether the last array identifier encountered was a single or double character (thus the scan mode may change in the middle of a line).

In Free-Format scan mode the lines are read character-by-character, so the subfields I, C and V need not be aligned to any particular columns. The following rules must be observed.

- 1) Columns 73-80 are ignored.
- 2) The I and C fields must not be separated by blanks.
- 3) The V field must be followed by a blank (or column 73).
- 4) Numbers must not have embedded blanks (e.g., in their exponents).
- 5) Commas are treated like blanks, i.e. they are delimiters.
- 6) The +, -, and & codes are not recognizable.
- 7) No more than 9 digits can be entered in sequence (in either the fraction or the exponent of a number).
- 8) The "V" field must be given for the codes, A, F, I, J, K, L, M, N, P, Q, R, and "blank" (i.e., where a V is ordinarily used).
- 9) The "V" field must not be given for the codes B, C, D, E, G, H, O, S, T, W, X, Y, Z, and \$, *, @, and % (i.e., where a V is ignored if present in fixed format).

In Fixed-Format scan mode the characters are interpreted as if by a standard (6(I2, A1, E9.0)) format. Embedded blanks are taken to be zeros, except that a full field of 12 blanks is ignored. Attempts to read Free-Format data with a Fixed-Format scan or vice-versa are generally unsuccessful, with AREAD becoming so confused that inappropriate error messages are printed.

When the % code is used, extra lines are read with a user-specified format (do not forget the outer parentheses on the variable-format card). This option makes it very easy to pick up data punched by other programs (although the power of the data option codes is then not available). When the @ code is used, a FORTRAN "unformatted READ" statement is executed. This allows easy access to intermediate data from other programs without going to the trouble and expense of punching cards.

If a B code is encountered, the next line is read immediately with a (20A4) format, and then the remainder of the line with the B is processed.

III. SUBROUTINE USER'S GUIDE

Variable Dimensioning

The array-oriented input scheme used by AREAD is strongly coupled to a method of variable dimensioning that is easy to implement in FORTRAN. This method relies upon the fact that a subroutine with an array argument need be given only the dimensions of the array and the address of its first element; the locations which are used for the array may actually be (in the calling program) a block of sufficient size starting at an arbitrary position in a large buffer. The programming required is most easily explained by example: A subroutine requires two arrays of variable size-- $H(I, J)$ real and $L(M, N)$ integer-- which are to be initialized by AREAD as the 2* and 3\$ arrays. A main program is required in which a buffer large enough to hold H and L is declared: $D(999)$ real equivalenced to $LD(999)$ integer. A list of pointers is placed in the buffer starting at location LONE: $LD(LONE + 1)$ is the index in D of the first element of H, $LD(LONE + 2) = LD(LONE + 1) + I*J$, $LD(LONE + 3) = LD(LONE + 2) + M*N$. Thus the 2* array is implicitly equivalenced to that section of D starting at $D(LD(LONE + 1))$ of length $I*J$, the 3\$ array starts at $LD(LD(LONE + 2))$, and the buffer must have a least $LD(LONE + 3)$ elements.

When AREAD is called and finds the 2* array identifier it starts loading real data into D at element $D(LD(LONE + 1))$ and expects to be given $(LD(LONE + 2) - LD(LONE + 1))$ elements. When the working subroutine is called, the real arguments are

(D(LONE + 1), D(LONE + 2), I, J, M, N) corresponding to the dummy parameters (H, L, I, J, M, N) with object time dimensions H(I, J) and L(M, N).

To summarize, D = LD is a buffer into which all the arrays are implicitly equivalenced. The starting point of each array is adjacent to the end of the previous one, so as to require the smallest possible buffer. The array identifier numbers indicate the array position in this close-packed sequence: array #1 starts at D(LD(LONE)), array #n starts at D(LD(LONE - 1 + n)). The number of elements in array #n is equal to (LD(LONE + n) - LD(LONE - 1 + n)). Note that the buffer dimension cannot be variable, so that the program's total memory requirement cannot be changed at object time. The advantage of the scheme is that only the buffer dimension need be changed to correspond to the minimum total array requirement, while the variably-dimensioned arrays are placed automatically.

Subroutine Arguments (NIN, NOUT, D, INDEX, NDXTYP, MODE, NY,
NY, KCARD, NEDIT, NERR, NEOF)

On Entry

NIN   unit number for input.

NOUT   unit number for messages and optional card echo.

D or LD   buffer in which variably-dimensional arrays are located.

INDEX   if NDXTYP > 0, ignored on entry.
              if NDXTYP $\leq$ 0, the "single array" option is in effect:
              D(INDEX) is the first element of the only array.

NDXTYP   if NDXTYP > 0, LD(NDXTYP) is the first element of the
              #1 array.
              if NDXTYP $\leq$ 0, the "single array" option is in effect:
              For the * and \$ options the array number is ignored
              and the length is indeterminate; for the % and @
              options the array number is used as the length;
              The F and E options are ignored.

MODE   determines the mode of the initial card scan:
              Free-Format if |MODE| = 2, Fixed-Format if |MODE| = 1.
              If NDXTYP $\leq$ 0, MODE > 0 implies a real array while
              MODE < 0 implies an integer array, so an array
              identifier is optional.

NY   an 80-element buffer, ignored on entry.

KCARD   a sequence number to be incremented by one as each line
              is read.

NEDIT   cards are echoed onto unit NOUT if NEDIT $\geq$ 0.

NERR   ignored on entry.

NEOF   ignored on entry.

On Exit:

NIN   final value of the unit number for input; the initial value may have been changed by X or Y codes.

NOUT   unchanged on exit.

D or LD   updated by the arrays which were read.

INDEX   if NDXTYP > 0, INDEX = array number of the last array identifier processed.
                if NDXTYP ≤ 0, then D(INDEX - 1) was the last element processed in the single array.

NDXTYP   unchanged on exit.

MODE   indicates the mode of the final card scan:
                the sign is determined by the last * or \$ array processed; the magnitude reflects Free or Fixed scan of the last field.

NY   contains the last line processed, in (80A1) format.

KCARD   equal to its initial value plus the number of lines read.

NEDIT   indicates the final state of the echo flag:
                the sign of the initial value is reversed if the final echo state is opposite that at entry.

NERR   equals the number of errors encountered before returning

NEOF   equals 0 if return is due to the T code;
                equals 1 if return is due to an end-of-file.

FIDO

AREAD can be incorporated quite easily into existing programs which use its predecessor FID0. A dummy FIDO is written which accepts the calls for input and passes them through to AREAD. A suitable routine to do this follows:

```
SUBROUTINE FIDO (LL2, J3, N5, N6)
COMMON/DUMMY/D(1)
DIMENSION CARD (80)
DATA NEDIT /-1/, KCARD/0/, MODE/2/
CALL AREAD (N5, N6, D, INDEX, LL2, MODE, CARD, KCARD,
           NEDIT, J3, NEOF)
RETURN
END
```

IV. PROGRAM MODIFICATION GUIDE

General Methods

AREAD has been written completely in FORTRAN-IV for the IBM/360. It is felt that the few essential machine-dependent features have been isolated for easy adaptation to other systems. The latest version required approximately 10 seconds to compile on a 360/91, using 250K bytes of memory. AREAD itself only occupies about 12K bytes when compiled.

Lines are read by AREAD with an (80A1) format and then scanned character-by-character. An internal code number is computed for each character, and the I, C, and V subfields are determined according to the character sequences. Blanks and commas have the same code number and are used as delimiters. Digit strings are accumulated as integers, with the number of decimal places subtracted from the optional E-type exponent. Only the integer string portion is returned in the I subfield, but for the V subfield the exponent is applied by lookup in a table of powers of ten. Array identifiers will be recognized in any column regardless of the scan mode and will immediately reset the Free/Fixed indicator; on entering Fixed-Format mode the column pointer is moved ahead to the next twelve-column full field.

After decomposing a whole line into a series of fields, AREAD processes each one in turn. The internal code number of each C-subfield is used in a computed GOTO to access the appropriate section of coding; the definitions of each code character are established at this point. New or revised options should be easy to insert.

Machine Dependencies

AREAD is felt to be machine dependent in only two ways. (But it is probably inevitable that others will be found.) The first is trivial, and is due to the limited range of the exponent of a REAL variable on an IBM/360. The vector CENTIM is supposed to be a table of powers of ten from 10**-90 to 10***90, but the two largest and smallest values are not representable. The table should be extended for a machine with a greater exponent range.

A more serious problem is presented by the requirement that characters be converted to the internal AREAD code given below:

<u>char</u> - <u>code</u>	<u>char</u> - <u>code</u>	<u>char</u> - <u>code</u>	<u>char</u> - <u>code</u>
0 0	A 11	a 11	- 37
1 1	B 12	b 12	+ 38
2 2	C 13	c 13	& 39
3 3	D 14	d 14	, 40
4 4	E 15	e 15	\$ 41
5 5	F 16	f 16	* 42
6 6	G 17	g 17	% 43
7 7	H 18	h 18	@ 44
8 8	I 19	i 19	¢ 45
9 9	J 20	j 20	? 46
· 10	K 21	k 21	47
	L 22	l 22	¡ 48
	M 23	m 23	" 49
< 254	N 24	n 24	' 50
> 254	O 25	o 25	
= 254	P 26	p 26	blank 40
/ 254	Q 27	q 27	
# 254	R 28	r 28	
! 254	S 29	s 29	
; 254	T 30	t 30	all
: 254	U 31	u 31	others 255
(254	V 32	v 32	
) 254	W 33	w 33	
_ 254	X 34	x 34	
	Y 35	y 35	
	Z 36	z 36	

The purpose of this translation is to insulate AREAD from the "natural" character code, which varies among different computers. The scheme described below is suitable for the original IBM/360 AREAD, but will require revision or replacement for other machines.

The problem is to set the integer KX equal to the internal code number of the character which was read with (A1) format into the INTEGER*4 KX1. The EBCDIC code for the character is in the left byte of KX1, right-padded with blanks. KX1 is equivalenced to the low-order word of the REAL*8 KX3; when this is added to KX6 the character is shifted to the right-most byte and the blanks are truncated (by careful choice of the DATA values of KX3 and KX6). The low-order word of KX4, the sum, is equivalenced to the INTEGER*4 KX7, which now contains the EBCDIC value of the character in the right-most byte, left-padded with zeros, i.e. KX7 is an integer with the EBCDIC value. Finally KX7 is used as an index into a cross tabulation of internal codes versus EBCDIC, $KX = KTABLE(KX7)$. In sum: a) use machine tricks to convert the (A1) format to the natural code; then b) index a translation table. This should be an easy scheme to simulate.

ACKNOWLEDGMENTS

This work was supported by U. S. Atomic Energy Commission Contract AT(11-1)-3073 and by the Electric Power Research Institute Project RP-113. Use was made of computer facilities supported in part by National Science Foundation Grant NSF-GP579.

—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, express 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 to the extent that such employee or contractor prepares, handles or distributes, or provides access to, any information pursuant to his employment or contract with the Commission.