Multi-Dimension Arrays

Contributed by James Basile, Department of Computer Science, C. W. Post Center, Long Island University, Greenvale, New York

It is easy to write defining words to provide one (or even two) dimension arrays in FORTH via CREATE . . . DOES>. Nonetheless, this leaves one in the position of having to write a new defining word for each size array as well as each data type (cf. [BAS81]). The attached code provides a defining word for generalized arrays of any size or type (within reason, of course!). It is not optimal for either the one or two dimension case, but could easily be made to decide such matters if speed were of the essence. It also only works for at least two dimensions since I have a defining word VECTOR for the one dimension case. However, you could incorporate this case by placing a test before the DO LOOP on line 6 in SCR # 1106.

To define an array using the structure you start with the word DIM, followed by the size desired for each dimension (much like in FORTRAN or BASIC), followed by the defining word ARRAY and its name. Thus,

```
DIM  2 5 3  ARRAY BLIP
```

defines a three dimension array BLIP with a total size of 30 cells (2x5x3). The type of the array is declared anywhere before the word ARRAY in the definition. This is accomplished by simply setting the variable B/E (bytes/entry) to an appropriate value. I provided several sample type declaration words in SCR # 1104. The code as written allows you to set the type once for several successive definitions. The word ARRAY can also be made to reset B/E to a default size (e.g., 2 bytes) if that is more desirable by adding the code `2 B/E !` at the end of line 4, SCR # 1106.

The number of dimensions is calculated by #DIM which makes use of the stack pointer to calculate how many dimensions were placed on the stack between DIM and ARRAY. SP@ is a system word that returns the stack pointer to the stack.

The array cells themselves are addressed in column major form with the indices beginning at 0, so that in the array BLIP the elements are in the order 0, 0, 0 1, 0, 0 0, 1, 0 1, 1, 0 etc. Range checking is done on array references via ?OK. If this feature is not desired ?OK may be eliminated from the code. The address computation algorithm is easily obtained from any compiler text, e.g., [AHO77].

References

[AHO77] Aho, A. and Ullman, J. *Principles of Compiler Design.* Reading: Addison-Wesley, 1977.

[BAS81] Basile, J. "Vectored Data Structures and Arithmetic Operators." *Proc. 1981 FORML Conference.* San Carlos: FIG, 1981.

```
SCR # 1104

 0 ( MULTI-DIMENSION ARRAYS )
 1 CREATE B/E 2 ,
 2 : CHARACTER 1+ B/E ! ;    ( n CHARACTER sets type as string )
 3 : DOUBLE 4 B/E ! ;        ( sets type to double precision )
 4 : SINGLE 2 B/E ! ;        ( sets type to single precision )
 5 : BYTE 1 B/E ! ;          ( sets type to byte )
 6
 7 VARIABLE (DIM)  ( temp to determine # of dimensions )
 8 : DIM SP@ (DIM) ! ;  ( note: SP@ fetches current stack ptr )
 9 : #DIM  ( -- n ) SP@ (DIM) @ - -2 / ;
10
11
12
13
14
15

SCR # 1105

 0 ( ARRAY DEFINING WORD PRIMITIVES )
 1 : NCOUNT  ( addr -- addr+2,count )  DUP 2+ SWAP @ ;
 2
 3 ( calculate limits for DO ... LOOP )
 4 : LIMITS  ( n1, n2 -- n1+n2, n1 )  OVER + SWAP ;
 5
 6 ( compute a partial offset into array )
 7 : IOFFSET  ( ith index, a<i+1st limit>, #left -- ith offset )
 8    2 * LIMITS DO  I @ * 2 +LOOP ;
 9
10 ( check each index to see if it is in range )
11 : ?OK  ( a<ith limit>, ith index -- a<i+1st limit>, ith index )
12    >R  NCOUNT R@ > NOT ABORT" bad array index" R> ;
13
14
15

SCR # 1106

 0 ( DEFINING WORD )
 1 : ARRAY  ( usage: DIM  I1 ... Im ARRAY <name> )
 2    CREATE  #DIM DUP , B E @  ( factor for size )
 3       SWAP  0 DO  OVER * ( size )  SWAP  ( Ith limit ) ,  LOOP
 4       B E @ ( bytes cell ) ,     ALLOT ( reserve space )
 5    DOES>     0 ( offset )  SWAP  ( pfa )
 6       NCOUNT 1 DO  ROT ?OK  ( check index in range )
 7          OVER  RP@ 2@ -  IOFFSET  ( compute Ith dim offset )
 8          ROT +  SWAP  ( add it in )  LOOP
 9       ROT ?OK ( last index )  ROT + ( final offset in cells )
10       >R  NCOUNT  ( bytes cell )  R> * + ;
11 ( some examples: )  EXIT
12 DIM  2 2 4 ARRAY TOTALS
13 DIM  8 8 BYTE ARRAY CHECKER-BOARD
14 DIM  20 10  32 CHARACTER ARRAY NAMES
15 DIM  10 5 2  DOUBLE ARRAY MONEY
```