SECTION III
Arrays

An array (or matrix) is a set of variables which is known by one name. The individual elements of an array are specified by the addition of a subscript to the array name: for example, M(7) is the seventh element of array M.

Arrays have either one or two dimensions. A one-dimensional array consists of a single column of many rows. The elements are specified by a single subscript, indicating the row desired. Rows and columns are numbered starting with 1. A two-dimensional array consists of a specified number of rows and a specified number of columns organized into a table. For example, an array M of five rows and three columns can be represented as follows:

Rows \ Columns	1	2	3
1	M(1,1)	M(1,2)	M(1,3)
2	M(2,1)	M(2,2)	M(2,3)
3	M(3,1)	M(3,2)	M(3,3)
4	M(4,1)	M(4,2)	M(4,3)
5	M(5,1)	M(5,2)	M(5,3)

Each element of the array is specified by a pair of subscripts separated by commas; the first indicates the row and the second the column.

Every array in a BASIC/3000 program is defined in one of three ways:

- Through a DIM statement that specifies the array name, and the number of rows and columns.
- Through a type declaration that specifies the same information as DIM and also declares the array to contain a particular data type.
- Through usage—numeric arrays that are used but are not explicitly defined in a DIM or type statement have 10 rows if one-dimensional or 10 rows and 10 columns if two-dimensional.

The physical size of an array is the total number of elements originally allocated to it; the logical size is the current number of rows times the current number of columns. The physical size of an array cannot be changed during execution, but the logical size (that is, the number of rows and columns) can be changed with a REDIM statement so long as the physical size is not exceeded.

BASIC/3000 permits arrays of all numeric data types as well as one-dimensional string arrays. Remarks in this section refer to numeric arrays, unless otherwise noted. String arrays are described in section V.

This section describes DIM and REDIM as used for numeric arrays. In addition it describes special statements used for computation and manipulation of one- and two-dimensional arrays. All of these statements begin with the word MAT.

Whenever an array is referenced within a function or a call to an external procedure, it must be subscripted by * if one-dimensional, or *,* if two-dimensional. For example,

```
DIM A(10), B(4,10)

CALL SETVALUE (X,A(*),B(*,*) )
```

DIM Statement

The DIM statement is used to reserve storage for arrays and to set upper bounds on the number of elements in arrays. DIM statements may also be used with strings (see Section V).

Form

DIM variable(integer),variable(integer), . . .

where the *variable* is the array name, and the *integer* specifies the number of rows in a one-dimensional array.

DIM variable(integer,integer),variable(integer,integer), . . .

where the *variable* names a two-dimensional array, and the first *integer* specifies the number of rows in the array, the second *integer* the number of columns.

Rows and columns are numbered starting with 1. The overall array size is the number of elements. In a one-dimensional array it is identical to the number of rows; in a two-dimensional array it is the product of the rows and columns.

More than one array can be named in a DIM statement; they are separated by commas.

Explanation

The elements of an array are specified by subscripted variables. The values of the elements are undefined when the program begins. The number of elements in the array is defined by a DIM statement, a type statement, or by usage. The DIM statement can appear anywhere in a program and is not executed. If control transfers to a DIM statement, execution falls through to the next sequential statement.

Examples

```
10 DIM A[15],B[15,5],B1[2,10]
20 REM    A HAS 15 ROWS, ONE COLUMN
30 REM      B AND B1 ARE TWO-DIMENSIONAL ARRAYS
40 REM      B HAS 15 ROWS, 5 COLUMNS;B1 HAS 2 ROWS,10 COLUMNS
50 DIM C[5],C1[5,1],C2[1,5]
60 REM     C AND C1 HAVE THE SAME DIMENSIONS: 5 ROWS, 1 COLUMN
70 REM     C2 HAS 1 ROW, 5 COLUMNS
```

Note that the DIM statement for C1 in line 50 would be the same if it were C1(5).

REDIM Statement

The REDIM statement is used to vary the number of rows and columns in arrays. REDIM is also used with strings (see Section V).

Form

REDIM variable(integer expression),variable(integer expression), . . .

REDIM variable(integer expression,integer expression),
variable(integer expression,integer expression), . . .

REDIM is like DIM except that the rows and columns can be specified with *integer expressions.* The value of the expression must be positive.

When more than one array is specified in a REDIM expression, they are separated by commas.

Explanation

The variables in a REDIM statement must have been previously dimensioned either explicitly with a DIM or type statement, or implicitly through use. When using REDIM to redimension an array, the number of rows and columns can be changed as desired provided these two conditions are met:

- The number of dimensions must not be changed.
- The total number of elements (rows times columns) must not be increased beyond the physical size (original dimensions) of the array.

Any data elements whose subscripts are included in both the old and new dimensions retain their old values in the newly dimensioned array. New elements have undefined values.

Arrays may be implicitly redimensioned in MAT READ, MAT INPUT, and the MAT Initialization and MAT Operation statements.

Examples

```
100 DIM A[20],B[5,5]
120 FOR X=1 TO 20
130   A[X]=0
140 NEXT X
150 B[1,4]=100
160 PRINT (FOR A=1 TO 20,A[A]),B[1,4]
170 REDIM A[10],B[2,6]
180 PRINT (FOR A=1 TO 10,A[A]),B[1,4]
```

Each element in A is set to zero, then one element in B is set to 100, and the results are printed. After redimensioning, the results are again printed. Note that B(4,1) is not affected by REDIM since it is still within the bounds of the redimensioned array.

```
 Ø            Ø            Ø            Ø            Ø
 Ø            Ø            Ø            Ø            Ø
 Ø            Ø            Ø            Ø            Ø
 Ø            Ø            Ø            Ø            Ø
 1ØØ
 Ø            Ø            Ø            Ø            Ø
 Ø            Ø            Ø            Ø            Ø
 1ØØ
```

In the example below, array C is dimensioned by use to have 10 rows and 10 columns, and array C5 to have 10 rows. An element in each array is assigned a value which, when the arrays are redimensioned, are out of the bounds of the array. Other elements within the new bounds are then given values and printed.

```
 10 C[4,1]=99
 20 C5[10]=0
 30 REM   BOTH C AND C5 ARE DIMENSIONED BY USE
 40 REM   C WITH 10 ROWS AND 10 COLUMNS
 50 REM   C5 WITH 10 ROWS
 60 PRINT C[4,1]
 70 PRINT C5[10]
 80 REDIM C[2,12],C5[5]
 90 REM   C(4,1) AND C5(10) ARE NO LONGER DEFINED
100 C[2,1]=70
110 C5[5]=100
120 PRINT LIN(1),C[2,1],C5[5]
>RUN
 99
 Ø

 70              100
```

Storing Data in Arrays

There are several methods of assigning values to arrays. Individual elements can be assigned using the assignment statement:

```
10 LET A[5]=26
20 B[I,9]=N*4.5
```

In addition, individual elements can appear in INPUT and READ statements:

```
10 INPUT A[1],A[2],A[3]
20 READ B[12]
```

If embedded FOR loops are used, entire arrays can be filled element by element:

```
10 INPUT (FOR N=1 TO 5,A[N])
20 READ (FOR N=1 TO 5,(FOR M=1 TO 5,B[N,M]))
```

To simplify the use of arrays, the MAT INPUT and MAT READ statements are provided to fill entire arrays.

MAT READ/INPUT STATEMENTS

The MAT READ statement assigns values from DATA statements to entire arrays, row by row. If dimensions are specified, the array is given new logical dimensions. The MAT INPUT statement is identical to MAT READ except that the values are taken from the input device (e.g., terminal) as in an INPUT statement.

Form

MAT READ array, array, . . .

MAT INPUT array, array, . . .

each *array* is either an array name (A,B7, etc.) or an array name followed by new dimensions (A(5), B(5,J)). The dimensions can be expressions. The rules for assigning new dimensions are given in the description of REDIM.

Explanation

If an array is dimensioned in MAT INPUT or MAT READ, the new logical size (i.e., the total number of elements) must not be more than were originally allocated to the array, nor may the number of dimensions be altered.

If the array is a string array, only the number of elements can be changed by MAT INPUT or MAT READ. The size of the elements in the string cannot be changed. (See Section V for a description of strings and string arrays.)

None of the special extensions available with a simple INPUT, such as saving excess input, are allowed with MAT INPUT.

Examples

```
  10 DIM A[9],C[10,4]
  20 MAT READ A
  25 RESTORE
  30 MAT READ C[8,4]
  40 PRINT A[1],A[5],A[9]
  50 PRINT C[1,1],C[5,2],C[8,4]
1000 DATA 1,2,3,4,5,6,7,8,9,10
1010 DATA 10,9,8,7,6,5,4,3,2,1
1030 DATA 30,31,32,33,34,35,36,37,38,39
1040 DATA 40,41,42,43,44,45,46,47,48,49
>RUN
 1             5             9
 1             3             41
```

Three elements from each array are printed. Array C is redimensioned by MAT READ in line 30. Note that the RESTORE and DATA statements have the same functions with MAT READ as they do with READ.

In the next example, the MAT INPUT statement expects input from the user. Both arrays A and C are printed in their entirety using FOR loops as print items.

```
  10 DIM A[9],C[2,3]
  20 MAT INPUT A,C
  30 PRINT (FOR N=1 TO 9,A[N])
  40 PRINT (FOR N=1 TO 2,(FOR M=1 TO 3,C[N,M]))
>RUN
?9,8,7,6,5,4,3,2,1
??22,33,44,55,66,77
 9             8             7             6             5
 4             3             2             1
 22            33            44            55            66
 77
```

Printing Data from Arrays

The mechanisms for printing data from arrays are parallel to those used for filling arrays. Individual elements can be printed using PRINT:

```
100 PRINT A[1],A[2],A[3]
```

If embedded FOR loops are used, entire arrays can be printed element by element:

```
100 PRINT (FOR N=1 TO 15,A[N])
200 PRINT (FOR N=1 TO 15,(FOR M=1 TO 5,B[N,M]))
```

To simplify the use of arrays, the MAT PRINT statement is provided to print entire arrays. MAT PRINT is also available for printing string arrays (see Section V) and for printing arrays to files (see Section VIII). In addition, the length of the print line can be changed by using the MARGIN statement (Section VIII) together with the MAT PRINT or MAT PRINT # statements. To print arrays according to a customized format, see the MAT PRINT USING and MAT PRINT # USING statements described in Section IX.

MAT PRINT STATEMENT

The MAT PRINT statement allows the printing of one or more complete arrays in a single statement. The elements are printed row by row and can be spaced out in fields or packed together, as in the PRINT statement (Section II).

Form

The form of a MAT PRINT statement is:

MAT PRINT mat print item,mat print item, . . .

A *mat print item* is either an array name or special function (TAB,LIN,CTL, and SPA); items are separated by a comma or semicolon and the list is optionally terminated by a comma or semicolon. FOR loops are not allowed in MAT PRINT.

Explanation

Each row of each array is printed separately, with double spacing between rows. If a comma follows the array, each element starts in one of the consecutive divisions of the line (see "PRINT Statement," Section II). If a semicolon follows the array, the elements are printed packed together, as if each element were followed by a semicolon. If nothing follows the last array, a comma is assumed. All formatting is done according to the specifications under PRINT statement.

An undefined array element causes the program to terminate. A one-dimensional array is printed as a single row.

Examples

```
  10 DIM A[10],B[5,5],C[2,2]
  20 MAT READ A,B[3,5],C
  30 MAT PRINT A
  40 PRINT
  50 MAT PRINT B,LIN(1),C,LIN(1)
  60 MAT PRINT A;LIN(2),B;
1000 DATA 2.5,46.7,75,0,50.1,0,0,0,19.8,0
1010 DATA 1,2,3,4,5,6,7,8,9,10
1020 DATA 11,12,13,14,15,16,17,18,19,20
>RUN
 2.5            46.7           75             0              50.1
 0              0              0              19.8           0


 1              2              3              4              5

 6              7              8              9              10

 11             12             13             14             15


 16             17

 18             19



 2.5            46.7           75     0       50.1           0      0      0
 19.8           0


 1      2       3      4       5

 6      7       8      9       10

 11     12      13     14      15
```

Note the effect of the semicolons following A and B in the MAT PRINT statement, line 60, on the printed output. MAT READ in line 20 redimensions array B; redimensioning of arrays is not permitted in a MAT PRINT statement.

Initializing Arrays

Three special functions (ZER, CON, IDN) provide the means to initialize numeric arrays with certain values, and optionally to redimension the arrays.

Form

The forms of MAT initialize statements are:

MAT numeric array=function
MAT numeric array=function(dimension)

The allowable *functions* are ZER, CON, and IDN.

The (*dimension*) part is optional and consists of one or two integer expressions separated by a comma. It changes the logical size of the array.

Explanation

ZER sets all elements of the array to zero.

CON sets all elements of the array to one.

IDN assigns an identity array to the array specified. The identity array is all zeroes, except the major diagonal which is all ones. The major diagonal starts in the upper left corner. If the array is not square, ones are extended along the diagonal as far as possible.

If an array is redimensioned by ZER, CON or IDN, the new size cannot have more elements than the original size, nor can the number of dimensions be altered.

Examples

```
 10 DIM A[5,5]
 20 MAT A=ZER
 30 MAT PRINT A
>RUN
 0             0             0             0             0

 0             0             0             0             0

 0             0             0             0             0

 0             0             0             0             0

 0             0             0             0             0
```

Function ZER sets each element in array A to zero.

```
  10 DIM A[4,4]
  20 MAT A=CON(3,4)
  30 MAT PRINT A
>RUN
 1             1             1             1

 1             1             1             1

 1             1             1             1
```

MAT A=CON(3,4) redimensions array A to have 3 rows and 4 columns, and sets each element in the newly dimensioned array to 1.

```
  10 DIM A[5,5]
  20 MAT A=IDN(4,4)
  30 MAT PRINT A
>RUN
 1             0             0             0

 0             1             0             0

 0             0             1             0

 0             0             0             1
```

IDN(4,4) changes the dimensions of A to 4 rows by 4 columns and sets the major diagonal to 1, the remaining elements to zero. If the array is not square, the extra elements are set to zero:

```
  10 DIM A[5,5]
  20 MAT A=IDN(5,3)
  30 MAT PRINT A
>RUN
 1             0             0

 0             1             0

 0             0             1

 0             0             0

 0             0             0
```

Array Operations

This group of six statements provides functions which operate on one or more entire arrays:

MAT Copy statement

MAT Add/Subtract statement

MAT Multiply statement

MAT Inverse statement

MAT Transpose statement

MAT Scalar Multiply statement

The arrays named in each statement all must be the same numeric type (see Section IV, Variable Types).

ARRAY COPYING

The MAT Copy statement copies one array into another. The form is

MAT numeric array=numeric array

The array on the right is copied into the array on the left. The destination array must have at least as many elements as the source and the same number of dimensions. It is redimensioned to have the same number of rows and columns as the source.

Examples

```
  10 DIM A1[2,3],B2[3,2]
  20 MAT READ B2
  30 MAT A1=B2
  40 MAT PRINT A1
1000 DATA 2.5,46.7,75,0,50.1,0,0,0,19.8,0
>RUN
 2.5              46.7

 75               0

 50.1             0
```

ARRAY ADDITION/SUBTRACTION

The MAT Add/Subtract statement performs array addition or subtraction (element by element) upon arrays of identical logical size and assigns the result to another array. The form is

MAT numeric array=numeric array+numeric array

MAT numeric array=numeric array – numeric array

The resulting array is assigned to the array on the left, which is redimensioned as in MAT copy. Any or all of these arrays may be the same array.

Examples

```
  10 DIM B[2,2],A1[2,2],A2[2,2]
  20 MAT READ A1,A2
  30 MAT B=A1+A2
  40 MAT PRINT A1,LIN(2),A2,LIN(2),B
1010 DATA 1,2,3,4,5,6,7,8,9,10
>RUN
 1             2

 3             4



 5             6

 7             8



 6             8

 10            12
```

The values in arrays A1 and A2 are added to produce the values printed for array B. Using the same data, A1 is subtracted from A2 to produce the following results in B:

```
  10 DIM B[2,2],A1[2,2],A2[2,2]
  20 MAT READ A1,A2
  30 MAT B=A1-A2
  40 MAT PRINT B
1010 DATA 1,2,3,4,5,6,7,8,9,10
>RUN
-4            -4

-4            -4
```

ARRAY MULTIPLICATION

The MAT Multiply statement performs an array multiplication on an array of dimension *m* by *n* and an array of dimension *n* by *p*; that is, the number of columns in the first array must equal the number of rows in the second. The result, a new array of dimension *m* by *p*, is assigned to a third array. The form is

*MAT numeric array = numeric array * numeric array*

Each row of the array to the left of * is multiplied by each column of the array on the right to produce the new element. The resulting array is assigned to the array to the left of the assignment operator. This array is redimensioned to dimension *m* by *p* as in the MAT Copy statement. Any or all of these arrays may be the same array.

Examples

```
 10 DIM A1[2,3],A2[3,2],B[2,2]
 20 MAT READ A1,A2
 30 MAT B=A1*A2
 40 MAT PRINT A1;LIN(1),A2;LIN(1),B;
100 DATA 1,2,3,4,5,6
200 DATA 4,5,6,7,8,9
>RUN
 1         2         3
 4         5         6

 4         5
 6         7
 8         9

 40        46
 94        109
```

array A1

array A2

*array B = A1*A2*

The method for performing a matrix multiplication is to multiply each element of the first row of array A1 by the corresponding element of the first column of A2 and to add the products. The result is the element B(1,1). Then each element in the first row of A1 is multiplied by the corresponding element in the second column of A2 and these are added to produce B(1,2). B(2,1) is the sum of the products resulting from the multiplication of row 2 of A1 and column 1 of A2; B(2,2) is the sum of the products of row 2 of A1 and column 2 of A2. To illustrate:

1X4 (4) + 2X6 (12) + 3X8 (24) = 40

4X4 (16) + 5X6 (30) + 6X8 (48) = 94

1X5 (5) + 2X7 (14) + 3X9 (27) = 46

4X5 (20) + 5X7 (35) + 6X9 (54) = 109

A second example multiplies the square array C by itself. In this case, the number of columns always equals the number of rows.

```
  10 DIM C[3,3]
  20 MAT INPUT C
  30 MAT PRINT C;LIN(1)
  40 MAT C=C*C
  50 MAT PRINT C;
>RUN
?2,4,6,8,1,3,5,7,9
 2      4      6

 8      1      3

 5      7      9


 66     54     78

 39     54     78

 111    90     132
```

To achieve the result MAT C=C*C;

C(1,1) = 2X2 (4) + 4X8 (32) + 6X5 (30) = 66

C(1,2) = 2X4 (8) + 4X1 (4) + 6X7 (42) = 54

C(1,3) = 2X6 (12) + 4X3 (12) + 6X9 (54) = 78

C(2,1) = 8X2 (16) + 1X8 (8) + 3X5 (15) = 39

C(2,2) = 8X4 (32) + 1X1 (1) + 3X7 (21) = 54

C(2,3) = 8X6 (48) + 1X3 (3) + 3X9 (27) = 78

C(3,1) = 5X2 (10) + 7X8 (56) + 9X5 (45) = 111

C(3,2) 5X4 (20) + 7X1 (7) + 9X7 (63) = 90

C(3,3) 5X6 (30) + 7X3 (21) + 9X9 (81) = 132

This next example multiplies a two-dimensional array with three rows and two columns by a one dimensional array with two rows. The result is a one-dimensional array with three rows.

```
 10 DIM A[3,2],B[2], C[3]
 20 MAT READ A
 30 MAT READ B
 40 MAT C=A*B
 50 DATA 1,2,3,4,5,6,1,2
 60 MAT PRINT A;LIN(1),B;LIN(1),C;
>RUN
 1         2

 3         4

 5         6


 1         2


 5         11        17
```

To achieve the result MAT C=A*B:

C(1)= 1X1 (1) + 2X2 (4) = 5

C(2)= 3X1 (3) + 4X2 (8) = 11

C(3)= 5X1 (5) + 6X2 (12) = 17

ARRAY INVERSION

The MAT Inverse statement assigns the inverse of a square array (i.e., number of rows equals number of columns) to another array. The inverse of an array is the array which, when multiplied by the original array, results in the identity array. The form is

MAT numeric array = INV (*numeric array*)

The arrays must not have been declared type integer (see Section IV). The array to the left of the assignment operator is redimensioned as in MAT Copy. The two arrays may be the same.

Example

```
  10 DIM A[10,3],B[5,5]
  20 MAT INPUT B
  30 MAT A=INV(B)
  40 MAT PRINT B,LIN(2),A
>RUN
?1,0,0,0,0,2,1,0,0,0,3,2,1,0,0,4,3,2,1,0,5,4,3,2,1
 1                 0                 0                 0                 0

 2                 1                 0                 0                 0

 3                 2                 1                 0                 0

 4                 3                 2                 1                 0

 5                 4                 3                 2                 1


 1                 0                 0                 0                 0

-2                 1                 0                 0                 0

 1                -2                 1                 0                 0

 0                 1                -2                 1                 0

 0                 0                 1                -2                 1
```

25 values are input to the square array B, then using INV, array A is set to the inverse of B. Array A is redimensioned to the same dimensions as B.

ARRAY TRANSPOSITION

The MAT Transpose statement assigns the transposition of an n by m array to an m by n array. Transposition switches rows and columns. The form is

MAT numeric array = TRN (*numeric array*)

The array to the left is redimensioned as in the MAT Copy statement. The two arrays may be the same.

Example

```
  10 DIM A[5,3],B[3,5]
  20 MAT INPUT B
  30 MAT A=TRN(B)
  40 MAT PRINT B,LIN(2),A
>RUN
?1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
 1             2             3             4             5

 6             7             8             9             10

 11            12            13            14            15


 1             6             11

 2             7             12

 3             8             13

 4             9             14

 5             10            15
```

Array A is the result of transposing array B with the TRN function. The columns in B are the rows in A; the rows in B are the columns in A.

ARRAY SCALAR MULTIPLICATION

The MAT Scalar Multiply statement multiplies all of the elements of an array by a specified value and assigns the result to another array. The form is

MAT *numeric array* = (*numeric expression*) * *numeric array*

The array to the left is redimensioned as in the MAT Copy statement. The two arrays may be the same.

Example

```
  10 N=5
  20 MAT INPUT B
  30 PRINT
  40 MAT A1=(N*2)*B
  50 MAT PRINT A1;
  60 DIM A1[3,4]
  70 DIM B[2,6]
>RUN
?1,2,3,4,5,6,7,8,9,10,11,12

 10     20     30     40     50     60

 70     80     90     100    110    120
```

Scalar multiplication simply multiplies each element of the array by the specified numeric expression, in this case N*2 or 10 since N=5. Each element of the resulting array A1 is 10 times the corresponding element in B. The dimensions of A1 are copied from B. The two arrays must be the same numeric type; the numeric expression may be a different type.

The numeric expression, if a different type, is converted to the type of the arrays before multiplication is performed. The conversion may affect the result if, for instance, the scalar expression is type real and the arrays are type integer. Consider the following:

MAT A = (2.5)*B, where A and B are integers, is equivalent to MAT A = (3)*B.

Array Functions

Two functions which can be used in expressions return information about arrays: ROW and COL.

ROW Function

The ROW function has the form

ROW (array)

and returns the number of rows in the array (a one-dimensional array consists of one column, many rows).

COL Function

The COL function has the form

COL (array)

and returns the number of columns in the array (returns 1 if the array is one-dimensional).

Examples

```
  10 READ M,N
  20 MAT READ A[M,N]
  30 PRINT (FOR I=1 TO ROW(A),(FOR J=1 TO COL(A),A[I,J]))
  40 DATA 3,5,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
>RUN
 1             2             3             4             5
 6             7             8             9             10
 11            12            13            14            15
```

The dimensions of array A are read into the variables M and N. The functions ROW(A) and COL(A) are used in the FOR loop to determine the print bounds for printing the array.