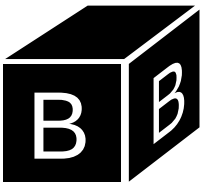


Chapter *18*

Two-Dimensional Arrays

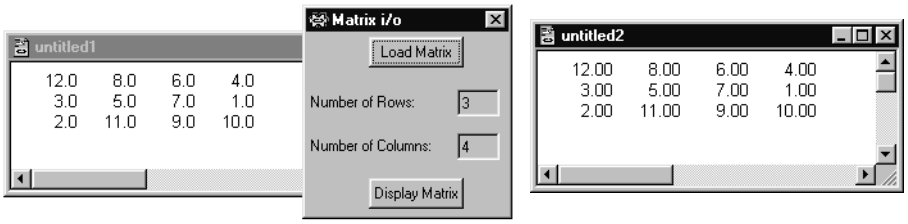


Figure 18.1
Input and output for a two-dimensional array of real values.

```
MODULE Pbox18A;
    IMPORT Dialog, TextModels, TextViews, Views, TextControllers, PboxMappers;
    TYPE
        Matrix = ARRAY 32, 32 OF REAL;
    VAR
        d*: RECORD
            numRows-, numCols-: INTEGER;
        END;
        matrix: Matrix;
```

Figure 18.2
A procedure that inputs a matrix and outputs it to a new window.

```
PROCEDURE LoadMatrix*;
    VAR
        md: TextModels.Model;
        cn: TextControllers.Controller;
        sc: PboxMappers.Scanner;
BEGIN
    cn := TextControllers.Focus();
    IF cn # NIL THEN
        md := cn.text;
        sc.ConnectTo(md);
        sc.ScanRealMatrix(matrix, d.numRows, d.numCols);
        Dialog.Update(d)
    END;
END LoadMatrix;
```

```
    PROCEDURE DisplayMatrix*;
        VAR
            md: TextModels.Model;
            vw: TextViews.View;
            fm: PboxMappers.Formatter;
    BEGIN
        md := TextModels.dir.New();
        fm.ConnectTo(md);
        fm.WriteRealMatrix(matrix, d.numRows, d.numCols, 8, 2);
        vw := TextViews.dir.New(md);
        Views.OpenView(vw)
    END DisplayMatrix;

BEGIN
    d.numRows := 0; d.numCols := 0;
END Pbox18A.
```

matrix[0,0] 12.0	matrix[0, 1] 8.0	matrix[0, 2] 6.0	matrix[0, 3] 4.0
matrix[1, 0] 3.0	matrix[1, 1] 5.0	matrix[1, 2] 7.0	matrix[1, 3] 1.0
matrix[2, 0] 2.0	matrix[2, 1] 11.0	matrix[2, 2] 9.0	matrix[2, 3] 10.0

Figure 18.3
Indices for a two-dimensional array of real values.

```
PROCEDURE (VAR s: Scanner) ScanRealMatrix (OUT mat: ARRAY OF ARRAY OF REAL;
        OUT numR, numC: INTEGER), NEW
Pre
s is connected to a text model.    20
Sequences of characters scanned represent in-range real or integer values.    21
All nonempty rows have the same number of values.    22
Number of rows in text model <= LEN(mat, 0).    Index out of range.
Number of columns in text model <= LEN(mat, 1).    Index out of range.
Post
mat gets all the values scanned up to the end of the text model to which s is connected.
numR gets the number of rows scanned.
numC gets the number of columns scanned.
The values are stored  at v[0..numR - 1, 0..numC - 1].
```

PROCEDURE (VAR f: Formatter) **WriteRealMatrix** (IN mat: ARRAY OF ARRAY OF REAL;
numR, numC, minWidth, dec: INTEGER), NEW
Pre
f is connected to a text model. 20
numR <= LEN(mat, 0). Index out of range.
numC <= LEN(mat, 1). Index out of range.
Post
The first numR rows and numC columns of mat are written to the text model to which f
is connected, each with a field width of minWidth and dec places past the decimal point.
If minWidth is too small to contain a value of mat it expands to accommodate the value.

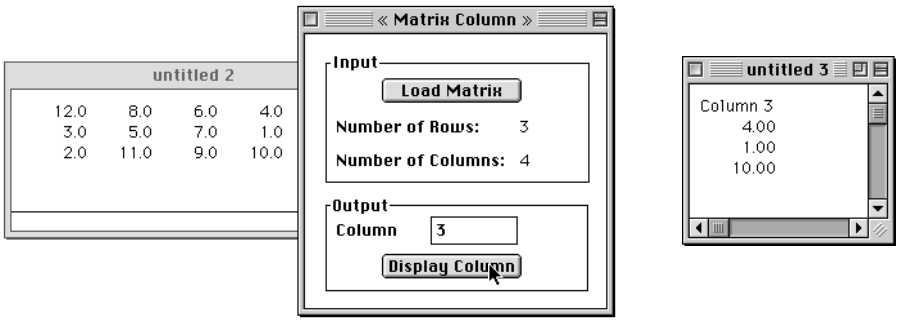


Figure 18.4
Printing a column of a matrix to a new window.

```
MODULE Pbox18B;
    IMPORT Dialog, TextModels, TextViews, Views, TextControllers, PboxMappers;
    TYPE
        Matrix = ARRAY 32, 32 OF REAL;
    VAR
        d*: RECORD
            numRows-, numCols-: INTEGER;
            column*: INTEGER;
        END;
        matrix: Matrix;

    PROCEDURE LoadMatrix*;
        VAR
            md: TextModels.Model;
            cn: TextControllers.Controller;
            sc: PboxMappers.Scanner;
    BEGIN
        cn := TextControllers.Focus();
        IF cn # NIL THEN
            md := cn.text;
            sc.ConnectTo(md);
            sc.ScanRealMatrix(matrix, d.numRows, d.numCols);
            Dialog.Update(d)
        END;
    END LoadMatrix;
```

Figure 18.5
A program to print a column of a matrix. The dialog is activated from a menu selection.

```
    PROCEDURE DisplayColumn*;
        VAR
            md: TextModels.Model;
            vw: TextViews.View;
            fm: PboxMappers.Formatter;
            i: INTEGER;
    BEGIN
        md := TextModels.dir.New();
        fm.ConnectTo(md);
        IF (0 <= d.column) & (d.column < d.numCols) THEN
            fm.WriteString("Column "); fm.WriteInt(d.column, 1); fm.WriteLn;
            FOR i := 0 TO d.numRows - 1 DO
                fm.WriteReal(matrix[i, d.column], 8, 2); fm.WriteLn
            END
        ELSE
            fm.WriteString("That column is not in the array")
        END;
        vw := TextViews.dir.New(md);
        Views.OpenView(vw)
    END DisplayColumn;

BEGIN
    d.numRows := 0; d.numCols := 0; d.column := 0;
END Pbox18B.
```

```
PROCEDURE MaxInRow (IN mat: ARRAY OF ARRAY OF REAL;
        row, numCols: INTEGER): REAL;
VAR
    max: REAL;
    j: INTEGER;
BEGIN
    ASSERT((0 <= row) & (row < LEN(mat, 0)), 20);
    ASSERT((0 < numCols) & (numCols <= LEN(mat, 1)), 21);
```

Figure 18.6
A function procedure that returns the maximum value in a row of a matrix.

```
PROCEDURE MaxInRow (IN mat: ARRAY OF ARRAY OF REAL;
        row, numCols: INTEGER): REAL;
VAR
    max: REAL;
    j: INTEGER;
BEGIN
    ASSERT((0 <= row) & (row < LEN(mat, 0)), 20);
    ASSERT((0 < numCols) & (numCols <= LEN(mat, 1)), 21);
    max := mat[row, 0];
    FOR j := 1 TO numCols - 1 DO
        IF mat[row, j] > max THEN
            max := mat[row, j]
        END
    END;
    RETURN max
END MaxInRow;
```

a

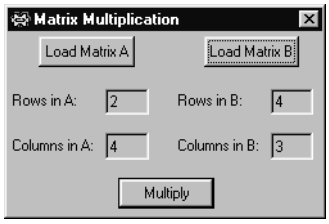
1.0	3.0	–1.0	2.0
0.0	4.0	1.0	–2.0

b

–1.0	2.0	5.0
–3.0	0.0	4.0
1.0	–2.0	3.0
3.0	2.0	–4.0

c

–5.00	8.00	6.00
–17.00	–6.00	27.00



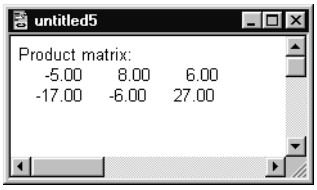


Figure 18.7
Output for the product of two matrices.

```
MODULE Pbox18C;
   IMPORT Dialog,  TextModels, TextViews, Views, TextControllers, PboxMappers;
   VAR
      d*: RECORD
         numRowA-, numColA-, numRowB-, numColB-: INTEGER;
      END;
      matrixA, matrixB: ARRAY 32, 32 OF REAL;

   PROCEDURE LoadA*;
      VAR
         md: TextModels.Model;
         cn: TextControllers.Controller;
         sc: PboxMappers.Scanner;
   BEGIN
      cn := TextControllers.Focus();
      IF cn # NIL THEN
         md := cn.text;
         sc.ConnectTo(md);
         sc.ScanRealMatrix(matrixA, d.numRowA, d.numColA);
         Dialog.Update(d)
      END
   END LoadA;
```

Figure 18.8
A program that multiplies two matrices.

```
PROCEDURE LoadB*;
    VAR
        md: TextModels.Model;
        cn: TextControllers.Controller;
        sc: PboxMappers.Scanner;
BEGIN
    cn := TextControllers.Focus();
    IF cn # NIL THEN
        md := cn.text;
        sc.ConnectTo(md);
        sc.ScanRealMatrix(matrixB, d.numRowB, d.numColB);
        Dialog.Update(d)
    END
END LoadB;
```

```
PROCEDURE Multiply (IN a, b: ARRAY OF ARRAY OF REAL;
        numRa, numCaRb, numCb: INTEGER;
        OUT c: ARRAY OF ARRAY OF REAL);
VAR
    sum: REAL;
    i, j, k: INTEGER;
BEGIN
    ASSERT((0 <= numRa) & (numRa <= LEN(a, 0)) & (numRa <= LEN(c, 0))
        & (0 <= numCaRb) & (numCaRb <= LEN(a, 1)) & (numCaRb <= LEN(b, 0))
        & (0 <= numCb) & (numCb <= LEN(b, 1)) & (numCb <= LEN(c, 1)), 20);
    FOR i := 0 TO numRa - 1 DO
        FOR j := 0 TO numCb - 1 DO
            sum := 0.0;
            FOR k := 0 TO numCaRb - 1 DO
                sum := sum + a[i, k] * b[k, j]
            END;
            c[i, j] := sum
        END
    END
END Multiply;
```

```
    PROCEDURE Compute*;
        VAR
            md: TextModels.Model;
            vw: TextViews.View;
            fm: PboxMappers.Formatter;
            matrixC: ARRAY 32, 32 OF REAL;
    BEGIN
        md := TextModels.dir.New();
        fm.ConnectTo(md);
        IF d.numColA = d.numRowB THEN
            Multiply(matrixA, matrixB, d.numRowA, d.numColA, d.numColB, matrixC);
            fm.WriteString("Product matrix:"); fm.WriteLn;
            fm.WriteRealMatrix(matrixC, d.numRowA, d.numColB, 8, 2)
        ELSE
            fm.WriteString("The number of columns in A must equal the number of rows in B.")
        END;
        vw := TextViews.dir.New(md);
        Views.OpenView(vw)
    END Compute;

BEGIN
    d.numRowA := 0; d.numColA := 0; d.numRowB := 0; d.numColB := 0;
END Pbox18C.
```

Statement	numRa **= 2**	numRa **= 3**	numRa **= *n***
(1)	3	4	$n + 1$
(2)	6	12	$(n + 1) \cdot n$
(3)	4	9	n^2
(4)	12	36	$n^2 \cdot (n + 1)$
(5)	8	27	n^3
(6)	4	9	n^2
Total:	37	97	$2n^3 + 4n^2 + 2n + 1$

Figure 18.9
Statement execution count for the procedure Multiply in Figure 18.8.

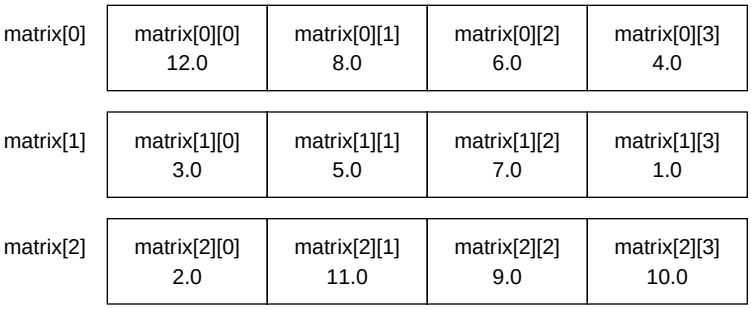


Figure 18.10
The matrix of Figure 18.3 considered as an array of vectors.