International Islamic University Chittagong
Department of Computer Science & Engineering
Autumn - 2022

Course Code: CSE-2321
Course Title: Data Structures

Mohammed Shamsul Alam
Professor, Dept. of CSE, IIUC

Lecture – 5

Multidimensional Arrays

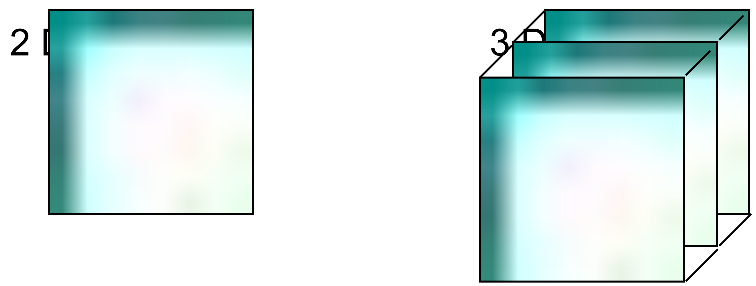
Multidimensional Arrays

A **multi-dimensional array** is an array of arrays.

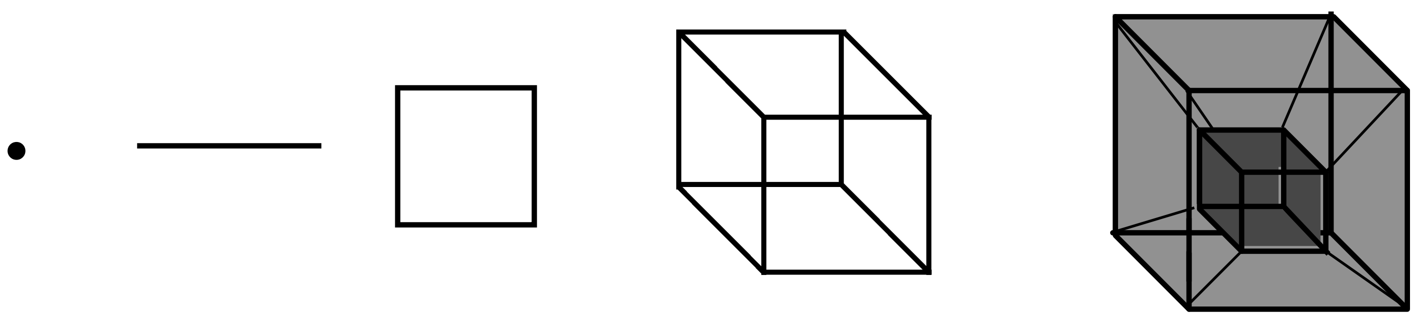
- 2-dimensional arrays are the most commonly used. They are used to store data in a tabular manner.
- Most programming languages allow two-dimensional and three-dimensional arrays, i.e., arrays where elements are referenced, respectively by two and three subscripts.
- In fact some programming languages allow the number of dimensions for an array to be as high as 7.
- In C programming an array can have two, three, or even ten or more dimensions. The maximum dimensions a C program can have depends on which compiler is being used.

Higher-Dimensional Arrays

An array can be declared with multiple dimensions.



Multiple dimensions get difficult to visualize graphically.



Two-Dimensional Arrays

A **two-dimensional** ***m x n*** array A is a collection of ***m . n*** data elements such that each element is specified by a pair of integers (such as **j** , **k**), called subscripts, with the property that

$$\mathbf{1 \le j \le m} \quad \text{and} \quad \mathbf{1 \le k \le n}.$$

- The element of A with first subscript j and second subscript k will be denoted by

$$\mathbf{A_{j,\,k}} \quad \text{or} \quad \mathbf{A\,[j, k]} \quad \text{or} \quad \mathbf{A\,[\,j\,]\,[\,k\,]}$$

- Two-dimensional arrays are called ***matrices*** in mathematics and ***tables*** in business applications; hence two-dimensional arrays are sometimes called matrix arrays.
- A standard way of drawing a two-dimensional *m x n* array A where the elements of A form a rectangular array with ***m rows*** and ***n columns*** and where the element A [j, k] appears in row j and column k. Fig- 4.8 [page-4.19] shows a 3 x 4 array.

Columns

$$\text{Rows}\quad \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & \quad 2 & \quad 3 & \quad 4 \end{array} \\ \begin{bmatrix} A[1,1] & A[1,2] & A[1,3] & A[1,4] \\ A[2,1] & A[2,2] & A[2,3] & A[2,4] \\ A[3,1] & A[3,2] & A[3,3] & A[3,4] \end{bmatrix} \end{array}$$

Fig. 4.8 *Two-Dimensional 3 × 4 Array A*

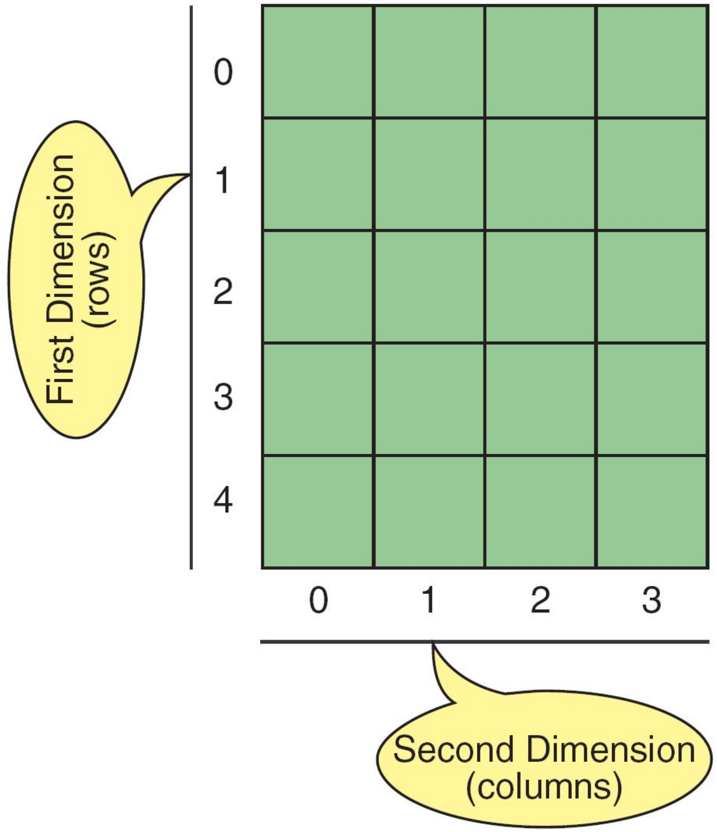


FIGURE 8-34 **A Two-dimensional Array (5 x 4)**

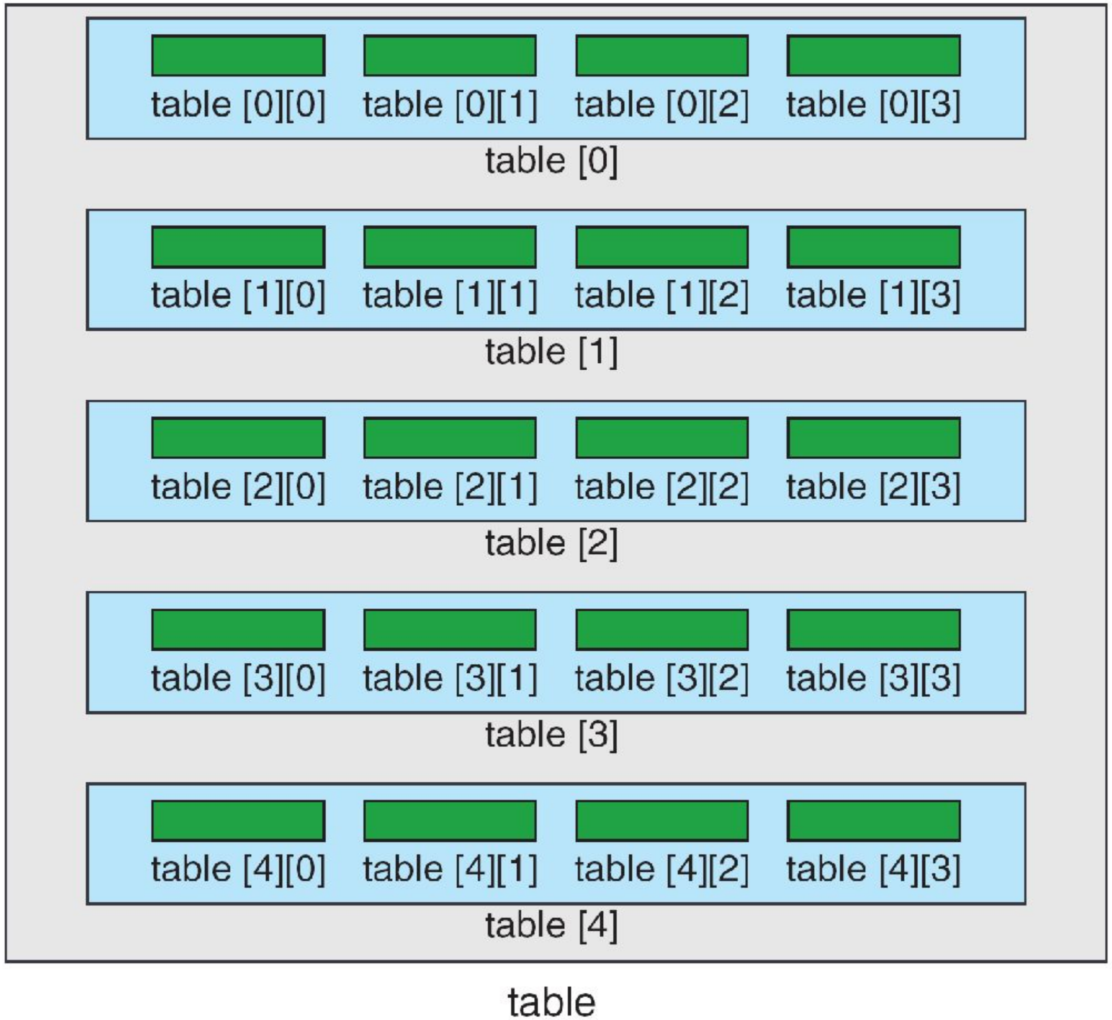


FIGURE C View of Two-dimensional Array

Representation of Two-Dimensional Arrays in Memory

Let A be a two-dimensional *m x n* array. Although A is pictured as a rectangular array of elements with m rows and n columns, the array will be represented in memory by a block of *m . n* sequential memory locations.

Specifically, the programming language will store the array A either

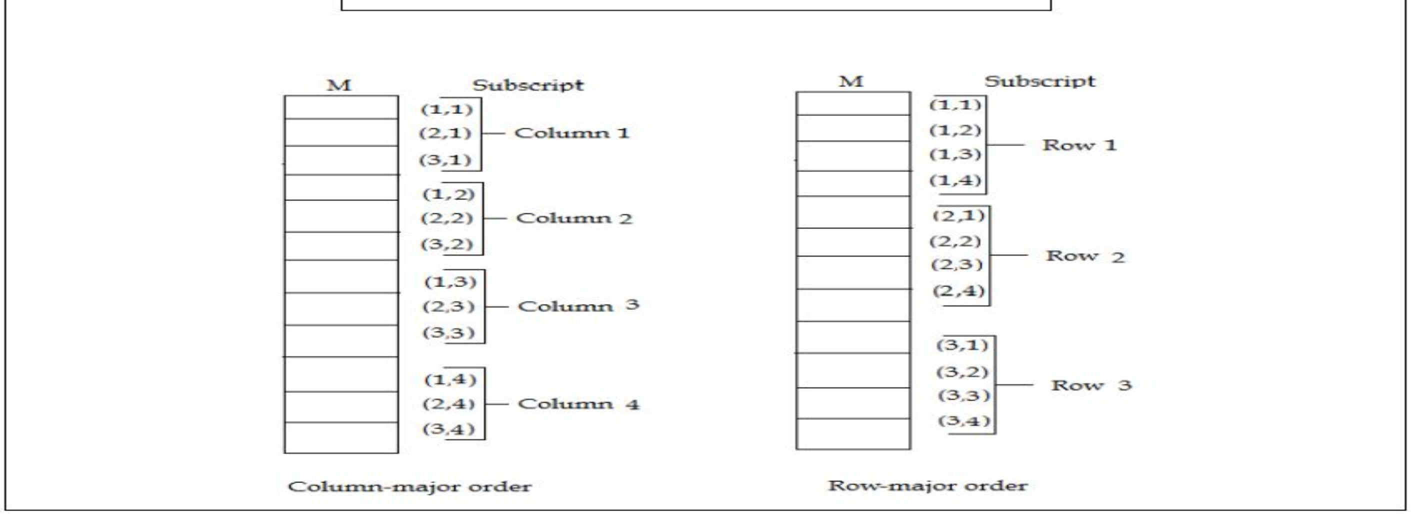
a) column by column, is what is called ***column-major order***, or

b) row by row, in ***row-major order***.

Row-major order is used in C,PL/I; column-major order is used in Fortran, MATLAB.

The following figure shows these two ways when A is a two-dimensional *3 x 4* array.

Figure 3.2: Two-dimensional 3 x 4 Array



Representation of Two-Dimensional Arrays in Memory

To compute the address LOC (A [J, K]) of A [J, K] we can use the following two formulas:

(Column-major order): **LOC (A [J, K]) = Base (A) + w [M (K - 1) + (J - I)]**

(Row-major order): **LOC (A [J, K]) = Base (A) + w [N (J - 1) + (K - 1)]**

Here Base (A) is the address of the first element A[1,1] of A.

Example: Consider that 25 students are given 4 tests. The students are numbered from 1-25 and the test score is assigned in a 25 x 4 matrix array - MARKS. Suppose Base (MARKS) = 100 and w = 4 bytes, and the program stores two dimensional arrays using row-major order.

The address of MARKS[10,2], that is the marks scored by the tenth student in the second test are as per the formula:

LOC (MARKS[J,K]) = Base(MARKS) + w [N(J-1) + (K-1)]

LOC (MARKS[10,2]) = 100 + 4 [4(10-1) + (2-1)]

= 100 + 4 [36+1] = 248

- If MARKS is stored using column-major order then what is the address of MARKS[10,2]?

2D Array Implementation in C

Write a program to read the elements of a 2D array and then display the elements of the array.

```
#include<stdio.h>
int main()
{
   int r, c, i, j, m[10][10];
   printf("How many rows and columns: ");
   scanf("%d%d",&r, &c);
   for(i=1 ; i<=r ; i++)
      for(j=1 ; j<=c ; j++)
         scanf("%d",&m[i][j]);
   printf("\nOutput:\n");
   for(i=1 ; i<=r ; i++)
   {
      for(j=1 ; j<=c ; j++)
         printf("%d\t",m[i][j]);
      printf("\n");
   }
   return 0;
}
```

General Multidimensional Arrays

General multidimensional arrays are defined analogously. More specifically, an n-dimensional $m_1 \times m_2 \times \dots \times m_n$ array B is a collection of $m_1 \cdot m_2 \dots m_n$ data elements in which each element is specified by a list of n integers—such as $K_1, K_2, \dots, K_n$—called *subscripts*, with the property that

$$1 \le K_1 \le m_1, \qquad 1 \le K_2 \le m_2, \qquad \dots, \qquad 1 \le K_n \le m_n$$

The element of B with subscripts $K_1, K_2, \dots, K_n$ will be denoted by

$$B_{K_1, K_2, \dots K_n} \qquad \text{or} \qquad B[K_1, K_2, \dots, K_N]$$

The array will be stored in memory in a sequence of memory locations. Specifically, the programming language will store the array B either in row-major order or in column-major order. By *row-major order*, we mean that the elements are listed so that the subscripts vary like an automobile odometer, i.e., so that the last subscript varies first (most rapidly), the next-to-last subscript varies second (less rapidly), and so on. By *column-major order*, we mean that the elements are listed so that the first subscript varies first (most rapidly), the second subscript second (less rapidly), and so on.

Example 4.13

Suppose B is a three-dimensional $2 \times 4 \times 3$ array. Then B contains $2 \cdot 4 \cdot 3 = 24$ elements. These 24 elements of B are usually pictured as in Fig. 4.11; i.e., they appear in three layers, called *pages*, where each page consists of the 2×4 rectangular array of elements with the same third subscript. (Thus the three subscripts of an element in a three-dimensional array are called, respectively, the *row*, *column* and *page* of the element.) The two ways of storing B in memory appear in Fig. 4.12. Observe that the arrows in Fig. 4.11 indicate the column-major order of the elements.

General Multidimensional Arrays

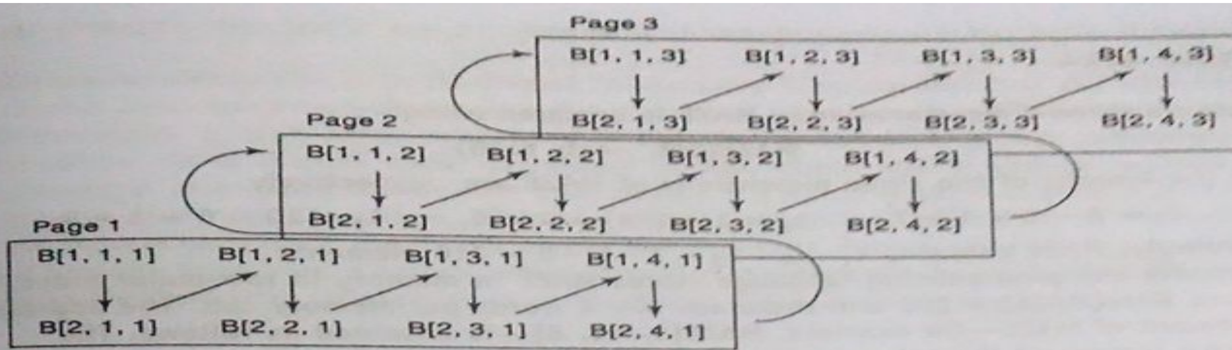
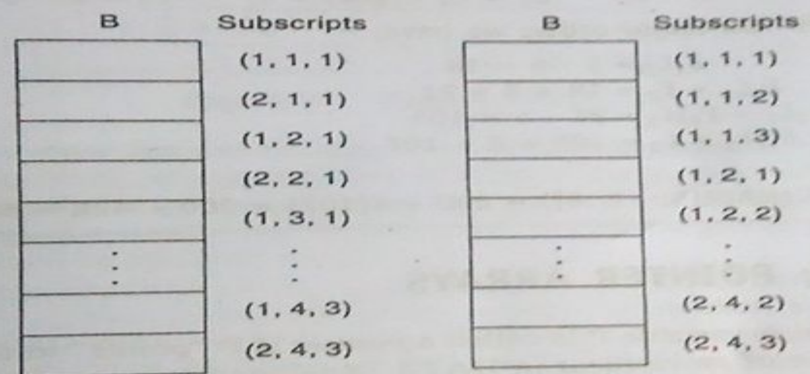


Fig. 4.11



(a) Column-major order (b) Row-major order

Fig. 4.12

For a given subscript K_i, the effective index E_i of L_i is the number of indices preceding K_i in the index set, and E_i can be calculated from

$$E_i = K_i - \text{lower bound} \tag{4.7}$$

Then the address LOC(C[K_1, K_2, ..., K_N] of an arbitrary element of C can be obtained from the formula

$$Base(C) + w[(((\ldots(E_N L_{N-1} + E_{N-1})L_{N-2}) + \ldots + E_3)L_2 + E_2)L_1 + E_1] \tag{4.8}$$

or from the formula

$$Base(C) + w[(\ldots((E_1 L_2 + E_2)L_3 + E_3)L_4 + \ldots + E_{N-1})L_N + E_N] \tag{4.9}$$

according to whether C is stored in column-major or row-major order. Once again, *Base*(C) denotes the address of the first element of C, and *w* denotes the number of words per memory location.

General Multidimensional Arrays

Example 4.14

Suppose a three-dimensional array MAZE is declared using

$$\text{MAZE}(2{:}8,\ -4{:}1,\ 6{:}10)$$

Then the lengths of the three dimensions of MAZE are, respectively,

$$L_1 = 8 - 2 + 1 = 7, \qquad L_2 = 1 - (-4) + 1 = 6, \qquad L_3 = 10 - 6 + 1 = 5$$

Accordingly, MAZE contains $L_1 \cdot L_2 \cdot L_3 = 7 \cdot 6 \cdot 5 = 210$ elements.

Suppose the programming language stores MAZE in memory in row-major order, and suppose *Base*(MAZE) = 200 and there are $w = 4$ words per memory cell. The address of an element of MAZE—for example, MAZE[5, −1, 8]—is obtained as follows. The effective indices of the subscripts are, respectively,

$$E_1 = 5 - 2 = 3, \qquad E_2 = -1 - (-4) = 3, \qquad E_3 = 8 - 6 = 2$$

Using Eq. (4.9) for row-major order, we have:

$$\begin{aligned} E_1L_2 &= 3 \cdot 6 = 18 \\ E_1L_2 + E_2 &= 18 + 3 = 21 \\ (E_1L_2 + E_2)L_3 &= 21 \cdot 5 = 105 \\ (E_1L_2 + E_3)L_3 + E_3 &= 105 + 2 = 107 \end{aligned}$$

Therefore,

$$\text{LOC}(\text{MAZE}[5, -1, 8]) = 200 + 4(107) = 200 + 428 = 628$$

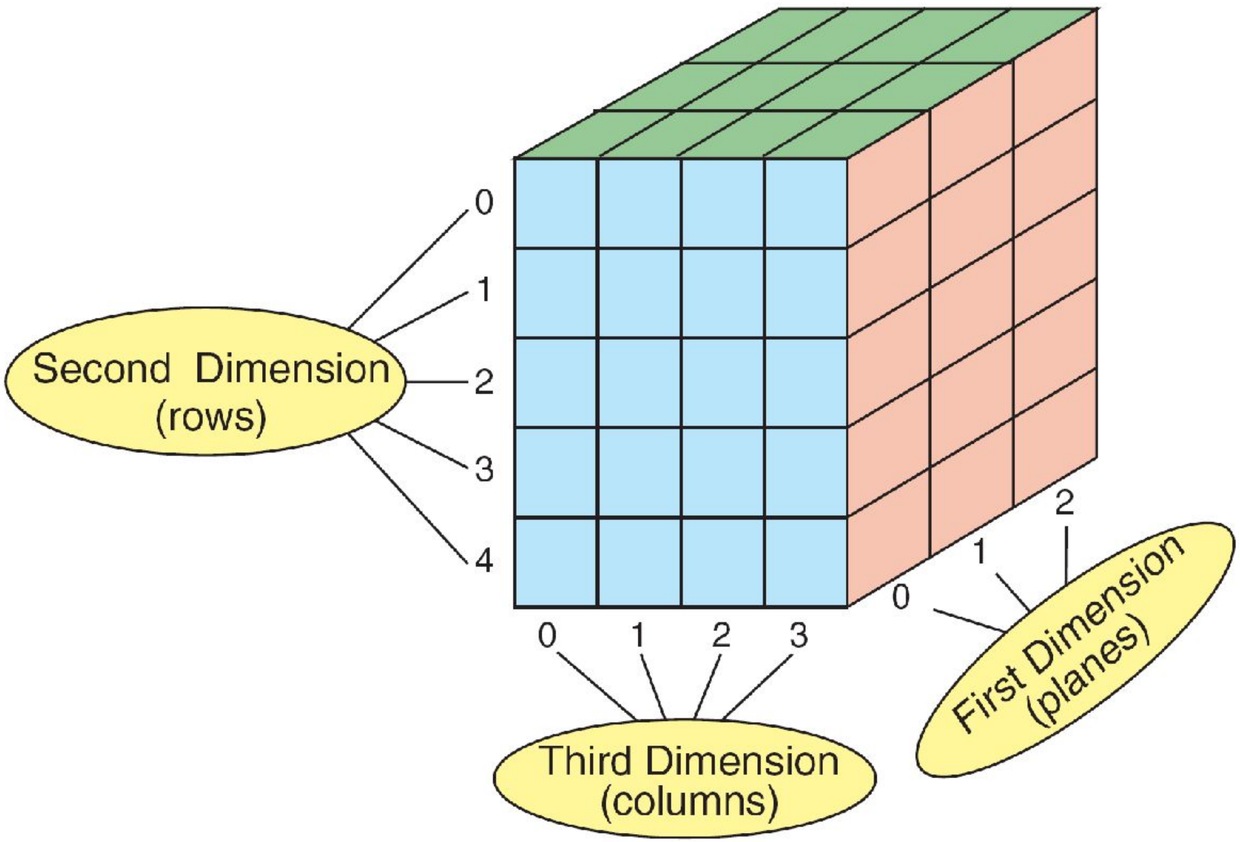


FIGURE 8-40 **A Three-dimensional Array (3 x 5 x 4)**

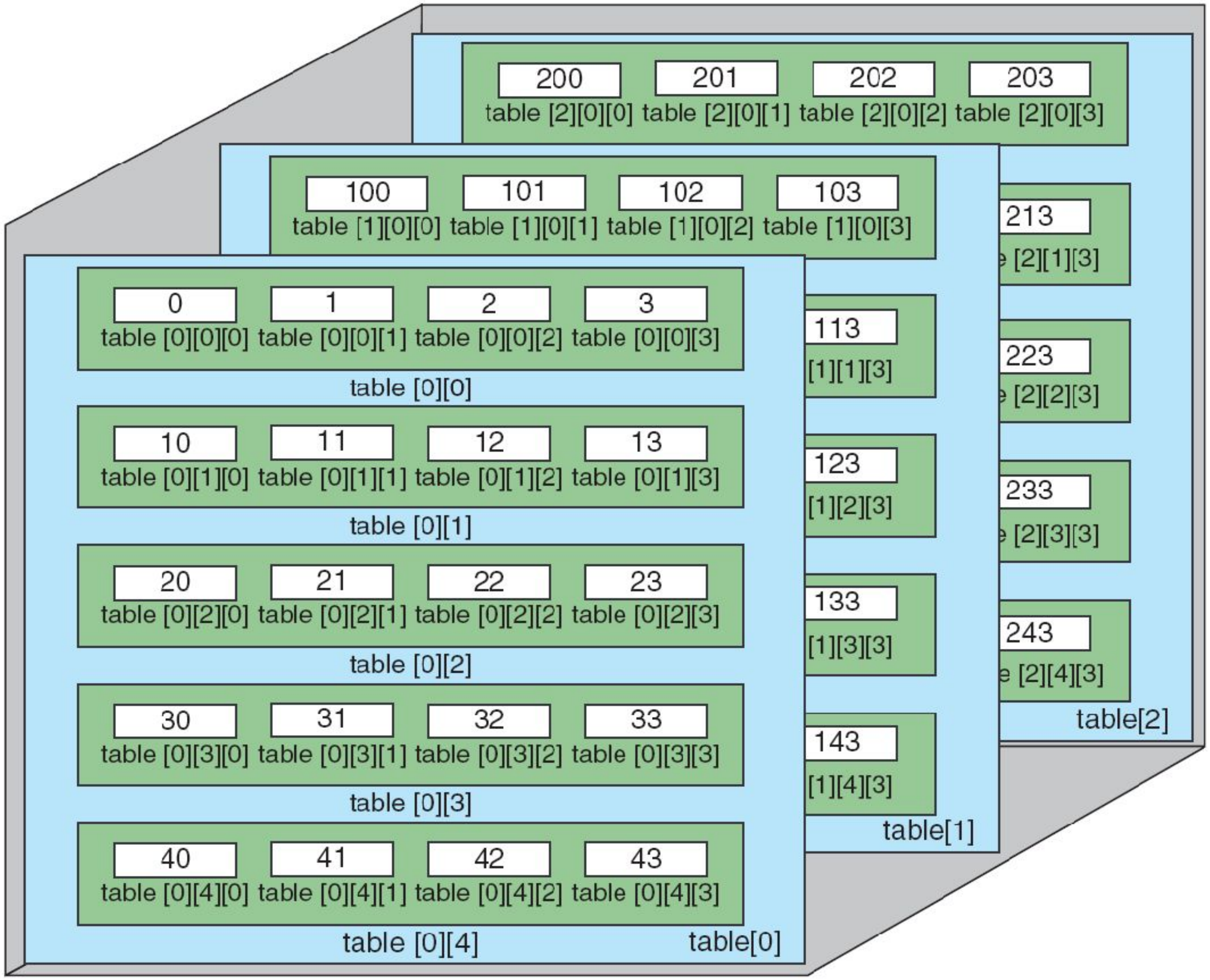


FIGURE 8-41 **C View of Three-dimensional Array**

Matrices

An *m x n* matrix A is an array of *m . n* numbers arranged in m rows and n columns as follows:

$$A = \begin{matrix} A_{11} & A_{12} & A_{13} & \ldots & A_{1n} \\ A_{21} & A_{22} & A_{23} & \ldots & A_{2n} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ A_{m1} & A_{m2} & A_{m3} & \ldots & A_{mn} \end{matrix}$$

- The **transpose** of a **matrix** is simply a flipped version of the original **matrix**. The result of transposing an *m x n* matrix is an *n x m* matrix with property:

  $M^T(j,i) = M(i,j)$, $1 <= i <= m$, $1 <= j <= n$.

- The sum of matrices is only defined for matrices that have **the same dimensions.** Suppose A and B are ***m x n*** matrices. The *sum* of A and B, written A + B, is the ***m x n*** matrix obtained by adding corresponding elements from A and B;

Suppose

$$A = \begin{pmatrix} 1 & -2 & 3 \\ 0 & 4 & 5 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 3 & 0 & -6 \\ 2 & -3 & 1 \end{pmatrix}$$

Then:

$$A + B = \begin{pmatrix} 1+3 & -2+0 & 3+(-6) \\ 0+2 & 4+(-3) & 5+1 \end{pmatrix} = \begin{pmatrix} 4 & -2 & -3 \\ 2 & 1 & 6 \end{pmatrix}$$

Matrix Multiplication

The product of matrices A and B is only defined when the **number of columns in A is equal to the number of rows in B**.

Suppose A is an ***m x p*** and suppose B is a ***p x n*** matrix. The product of A and B, written AB, is the ***m x n*** matrix C whose ij th element C_{ij} is given by

$$C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} + \dots + A_{ip}B_{pj} = \sum_{k=1}^{p} A_{ik}B_{kj}$$

(c) Suppose

$$A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 2 & 0 & -4 \\ 3 & 2 & 6 \end{pmatrix}$$

The product matrix AB is defined and is a 2×3 matrix. The elements in the first row of AB are obtained, respectively, by multiplying the first row of A by each of the columns of B:

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}\begin{pmatrix} 2 & 0 & -4 \\ 3 & 2 & 6 \end{pmatrix} = \begin{pmatrix} 1\cdot 2+3\cdot 3 & 1\cdot 0+3\cdot 2 & 1\cdot(-4)+3\cdot 6 \\ & & \end{pmatrix} = \begin{pmatrix} 11 & 6 & 14 \\ & & \end{pmatrix}$$

Similarly, the elements in the second row of AB are obtained, respectively, by multiplying the second row of A by each of the columns of B:

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}\begin{pmatrix} 2 & 0 & -4 \\ 3 & 2 & 6 \end{pmatrix} = \begin{pmatrix} 11 & 6 & 14 \\ 2\cdot 2+4\cdot 3 & 2\cdot 0+4\cdot 2 & 2\cdot(-4)+4\cdot 6 \end{pmatrix} = \begin{pmatrix} 11 & 6 & 14 \\ 16 & 8 & 16 \end{pmatrix}$$

That is,

$$AB = \begin{pmatrix} 11 & 6 & 14 \\ 16 & 8 & 16 \end{pmatrix}$$

Matrix Multiplication

(Matrix Multiplication) MATMUL (A, B, C, M, P, N)

Let A be an M x P matrix array, and let B be a P x N matrix array. This algorithm stores the product of A and B in an M x N matrix array C.

1. Repeat Steps 2 to 4 for I = 1 to M:
2. Repeat Steps 3 and 4 for J = 1 to N:
3. Set C [I, J] := 0
4. Repeat for K= 1 to P:
   C [I, J] := C [I, J] + A [I, K] * B [K, J]
5. Exit.

Sessional:

1. Write a program to interchange the row and column of a matrix.
2. Write a program to add two matrices.
3. Write a program to calculate the multiplication of two matrices.

Sparse Matrices

Matrices with a relatively ***high proportion of zero*** entries are called *sparse matrices*. If the number of zeros in a matrix exceeds (n*m)/2, where n, m is the dimension of the matrix, then the matrix is called **sparse matrix.**

- Two general types of n-square sparse matrices are there which occur in various applications are mention in figure below(It is sometimes customary to omit blo

$$\begin{pmatrix} 4 & & & & \\ 3 & -5 & & & \\ 1 & 0 & 6 & & \\ -7 & 8 & -1 & 3 & \\ 5 & -2 & 0 & 2 & -8 \end{pmatrix}$$

(a) Triangular matrix

$$\begin{pmatrix} 5 & -3 & & & & & \\ 1 & 4 & 3 & & & & \\ & 9 & -3 & 6 & & & \\ & & 2 & 4 & -7 & & \\ & & & 3 & -1 & 0 & \\ & & & & 6 & -5 & 8 \\ & & & & & 3 & -1 \end{pmatrix}$$

(b) Tridiagonal matrix

- Triangular matrix This is the matrix where all the entries above the main diagonal are zero or equivalently where non-zero entries can only occur on or below the main diagonal is called a (lower)Triangular matrix.
- Tridiagonal matrix This is the matrix where non-zero entries can only occur on the diagonal or on elements immediately above or below the diagonal is called a Tridiagonal matrix.
- The natural method of representing matrices in memory as two-dimensional arrays may not be suitable for sparse matrices i.e. one may save space by storing only those entries which may be non-zero

Sparse Matrices

Example 4.25

Suppose we want to place in memory the triangular array A in Fig. 4.22. Clearly it would be wasteful to store those entries above the main diagonal of A, since we know they are all zero; hence we store only the other entries of A in a linear array B as indicated by the arrows. That is, we let

$$B[1] = a_{11},\quad B[2] = a_{21},\quad B[3] = a_{22},\quad B[3] = a_{31},\quad \ldots$$

Observe first that B will contain only

$$1 + 2 + 3 + 4 + \ldots + n = \frac{1}{2}n(n + 1)$$

elements, which is about half as many elements as a two-dimensional $n \times n$ array. Since we will require the value of a_{JK} in our programs, we will want the formula that gives us the integer L in terms of J and K where

$$B[L] = a_{JK}$$

Observe that L represents the number of elements in the list up to and including a_{JK}. Now there are

$$1 + 2 + 3 + \ldots + (J - 1) = \frac{J(J-1)}{2}$$

elements in the rows above a_{JK}, and there are K elements in row J up to and including a_{JK}. Accordingly,

$$L = \frac{J(J-1)}{2} + K$$

yields the index that accesses the value a_{JK} from the linear array B.

Sparse Matrices

Home Task:

4.13 Consider an n-square tridiagonal array A as shown in Fig. 4.24. Note that A has n elements on the diagonal and $n-1$ elements above and $n-1$ elements below the diagonal. Hence A contains at most $3n-2$ nonzero elements. Suppose we want to store A in a linear array B as indicated by the arrows in Fig. 4.24; i.e.,

$$B[1] = a_{11}, \quad B[2] = a_{12}, \quad B[3] = a_{21}, \quad B[4] = a_{22}, \quad \ldots$$

Find the formula that will give us L in terms of J and K such that

$$B[L] = A[J, K]$$

(so that one can access the value of A[J, K] from the array B).

$$\begin{pmatrix} a_{11} \to a_{12} & & & \\ a_{21} \to a_{22} \to a_{23} & & & \\ & a_{32} \to a_{33} \to a_{34} & & \\ & & \cdots\cdots & \\ & & & a_{n,n-1} \to a_{nn} \end{pmatrix}$$

Fig. 4.24 *Tridiagonal Array*