Arrays

Defining arrays, declaration and initialization of arrays

Designed by Parul Khurana, LIECA.

Introduction

- Many applications require the processing of multiple data items that **have** common characteristics (e.g., a set of numerical data, represented by $x_1, x_2, x_3, \ldots x_n$).
- In such situations it is often convenient to place the data item into an *array*, where they all share the same name (e.g., x).
- The individual data items can be **characters, integers, floating-point numbers**, etc. However, they must all be of the same type and the same storage class.

Introduction

- Each array element (i.e., each individual data item) is referred to by specifying the array name followed by one or more subscripts, with each subscript enclosed in square brackets. Each subscript must be expressed as a non-negative integer.
- *In an n-element array, the array elements are x[0], x[1], ..., x[n-1], as illustrated in Fig.*
- 

Introduction

- The value of each subscript can be expressed as an integer constant or an integer constant variable or a more complex integer expression.
- The number of subscripts determines the **dimensionality** of the array. For example, x[i] refers to an element in the one dimensional array x. Similarly, y[i][j] refers to an element in the two-dimensional array y.

Defining Arrays

- Arrays are defined in much the same manner as ordinary variables, except that each array name must be accompanied by a size specification (i.e., the number of elements).
- For a one-dimensional array, the size is specified by a positive integer expression, enclosed in square brackets. The expression is usually written as a positive integer constant.

Defining Arrays – Method 1

- In general terms, a one-dimensional array definition may be expressed as:

```
storage-class    data-type    array [ expression ];
```

- where storage-class refers to the storage class (namely, auto, extern, static, register) of the array, data-type is the data type, array is the array name, and expression is a positive-valued integer expression which indicates the number of array elements.

Defining Arrays – Method 1

- In general terms, a one-dimensional array definition may be expressed as:

```
storage-class    data-type    array [ expression ];
```

- the storage-class in **optional**; default values are *automatic* for arrays that are defined within a function or a block and *external* for arrays that are defined outside of a function.

Defining Arrays - Examples

- Several typical one-dimensional array definitions are shown below:

```
int x[100];
char text [80];
static char message [25];
static float n [12];
```

- The first line states that x is a 100-element integer array.
- The second line defines text to be an 80-element character array.
- In the third line, message is defined as static 25-element character array, whereas the fourth line establishes n as static 12-element floating-point array.

Defining Arrays – Method 2

- It is sometimes convenient to define an array size in terms of a *symbolic constant* rather than a fixed integer quantity.
- This makes it easier to modify a program that utilizes an array, since all references to the maximum array size can be altered simply by changing the value of the symbolic constant.

Defining Arrays – Examples

- Typical one-dimensional array definition is shown below:

```
#define SIZE 100

int x[SIZE];
```

- Notice that the symbolic constant SIZE is assigned a value of 100. This *symbolic constant*, rather than its value, appears in the array definition.
- (*Remember that the value of the symbolic constant will be substituted for the constant itself by the* ***preprocessor*** *before the compilation [translation] process*.)

Initializing Arrays

- The general form is:

```
storage-class   data-type   array [ expression ] = { value 1, value 2, … , value n} ;
```

- where value 1 refers to the value of the first array element, value 2 refers to the value of the second element, and so on.
- The appearance of the expression, which indicates the number of array elements, is optional when initial values are present.

Initializing Arrays - Examples

- Shown below are several array definitions that include the assignment of initial values:

```
int digits[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
static float x [6] = { 0, 0.25, 0, -0.50, 0 , 0 };
char color [3] = {  'R', 'E', 'D' };
```

- Note that x is a static array.
- The other two arrays (digits and color) are assumed to be external arrays by virtue of their placement within the program.

Initializing Arrays - Note

- *All individual array elements that are not assigned explicit initial values will automatically be set to zero.*

Initializing Arrays – Examples- Variations

- Consider the following array definitions:

```
int digits[10] = { 3, 3, 3 };
```

- The result, on an element-by-element basis, are as follows: (*Remember that the subscripts in an n-element array range from 0 to n-1*).

- **All elements are set to 0 except those that have been explicitly initialized within array definitions.**

digits [0] = 3	digits [5] = 0
digits [1] = 3	digits [6] = 0
digits [2] = 3	digits [7] = 0
digits [3] = 0	digits [8] = 0
digits [4] = 0	digits [9] = 0

Initializing Arrays – Examples- Variations

- Consider the following array definitions:

```
int digits[ ] = { 1, 2, 3, 4, 5, 6 };
```

digits [0] = 1
digits [1] = 2
digits [2] = 3
digits [3] = 4
digits [4] = 5
digits [5] = 6

- Thus digits, will be a six-element integer array.
- **The array size need not be specified explicitly when initial values are included as a part of as array definition.**

Initializing Arrays

- Strings (i.e., character arrays) are handled somewhat differently.
- In particular, when a string constant is assigned to an external or a static character array as a part of the array definition, the array size specification is usually omitted.
- The proper array size will be assigned automatically.
- This will include a provision for the null character \0, which is automatically added at the end of every string.

Initializing Arrays – Examples- Variations

- Consider the following two character array definitions.
- Each includes the initial assignment of the string constant "RED".
- However ,the *first array is defined as a three – element array,* whereas the size of the second array is unspecified.

```
char color [3]  =  "RED";
char color [ ] = "RED";
```

- The results of these initial assignments are not the same because of the null character, \0, which is automatically added at the end of the second string.

Initializing Arrays – Examples- Variations

- Thus, the elements of the first array are:

```
char color [0]  =‘R’;
char color [1] = ‘E’;
char color [2] = ‘D’;
```

```
char color [3]  =  “RED”;
```

- Thus, the elements of the second array are:

```
char color [0]  =‘R’;
char color [1] = ‘E’;
char color [2] = ‘D’;
char color [3] = ‘\0’;
```

```
char color [ ] = “RED”;
```

Initializing Arrays – Examples- Variations

- Thus, the first form is incorrect, since the null character \0 is not included in the array.
- The array definition could also have been written as:

```
char color [4]  =  “RED”;
```

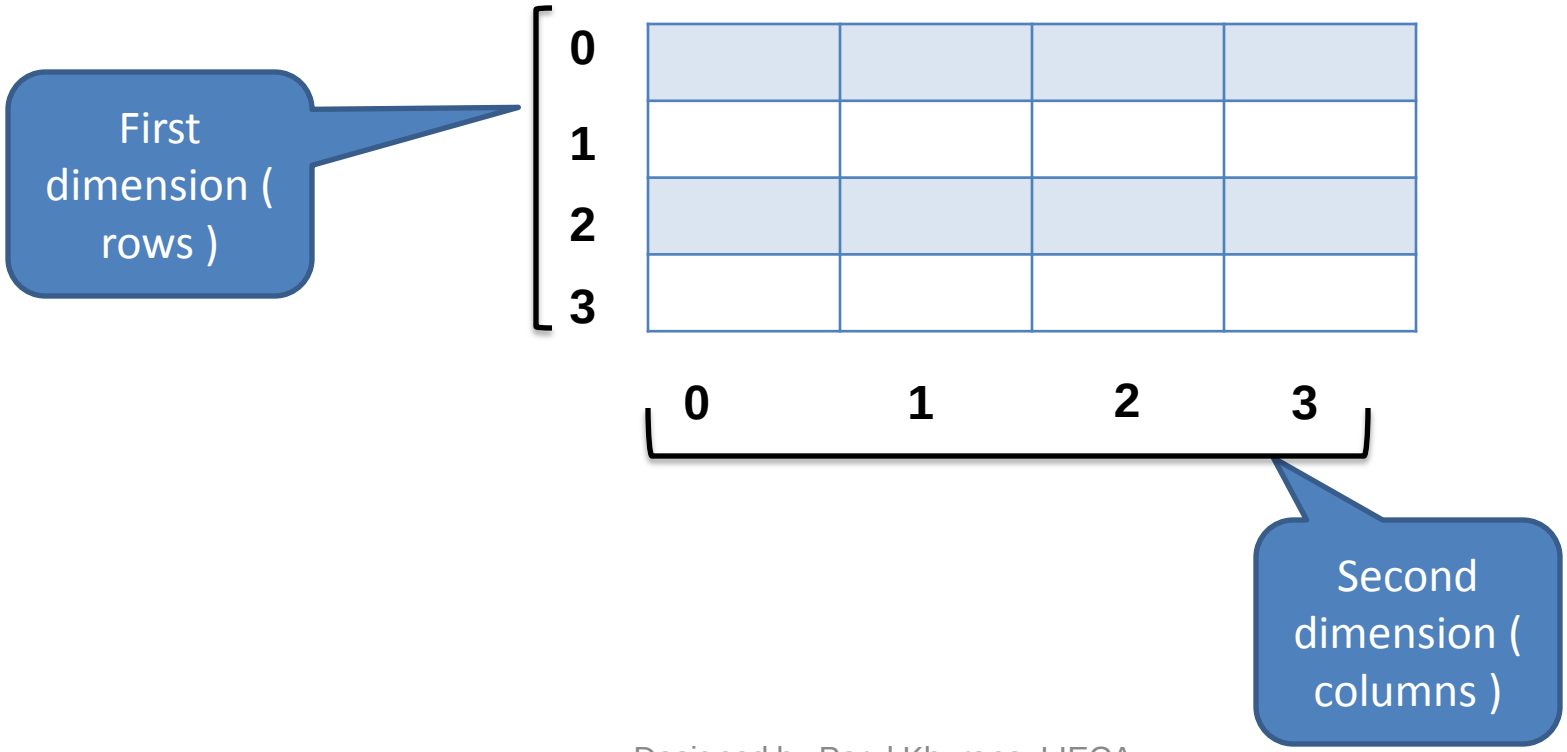
- *This definition is correct, since we are now defining a four-element array which includes an element for the null character.*

Defining Arrays – 2D

- The arrays we have discussed so far are known as One-dimensional arrays because the data are organized linearly only in one direction (dimension).
- Many applications require that the data be organized in more than one dimension.
- One common example is matrix (*a table*), which is an array that consists of rows and columns.

Defining Arrays – 2D

- Following Figure shows a matrix (table) with 4 rows and 4 columns which is commonly used as a two-dimensional array.



Defining Arrays – 2D

- Although two-dimensional array is exactly what is shown in previous Figure, C language looks at it in a different way.
- *It looks at the two-dimensional array as an array of arrays*.
- In other words, a two-dimensional array in C language is an array of one-dimensional arrays. This concept is shown in the next figure.

Defining Arrays – 2D

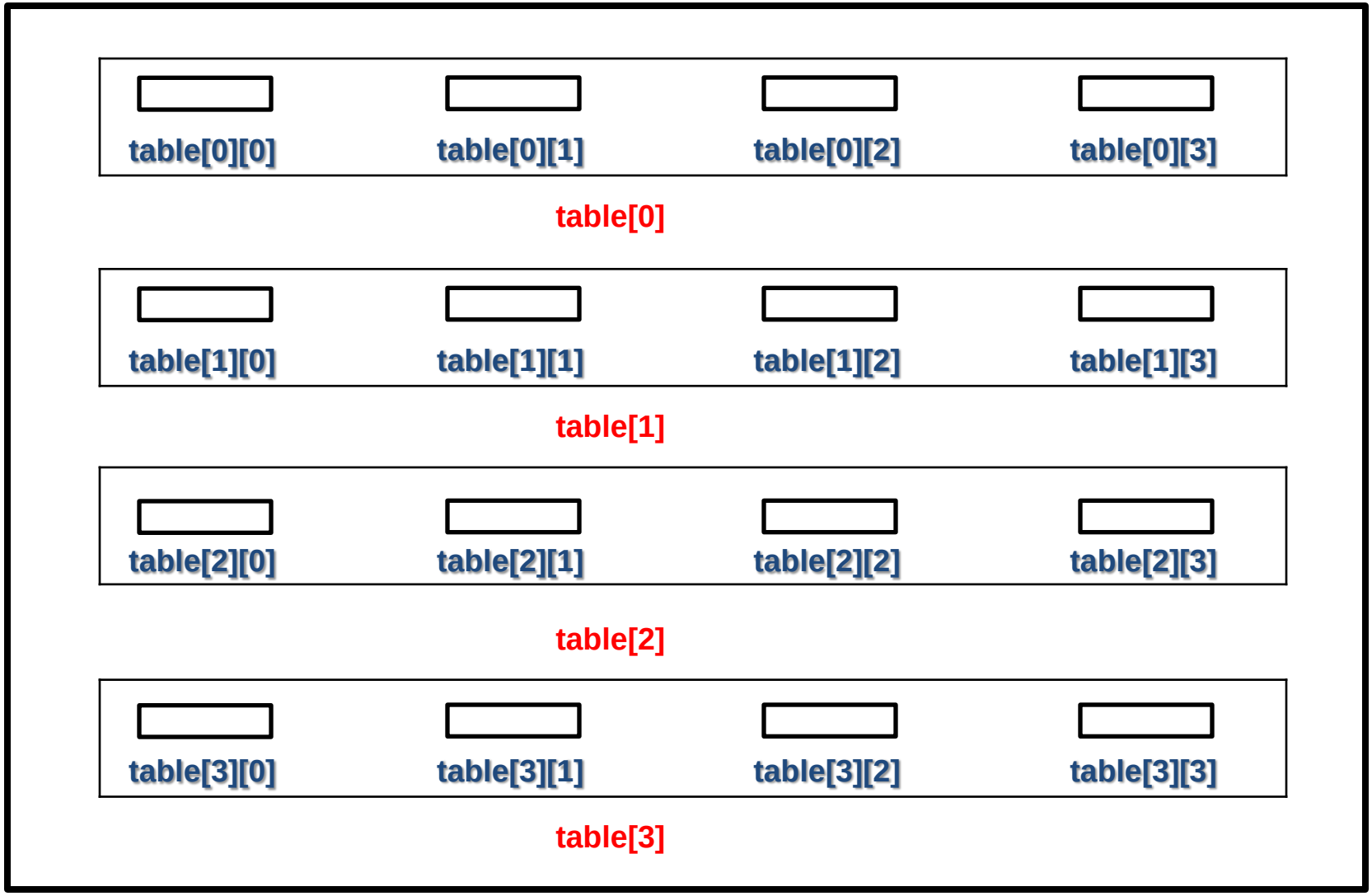


table **Array of arrays**

Defining Arrays – 2D

- In general terms, a two-dimensional array definition may be expressed as:

```
storage-class    data-type    array [ expression 1 ] [ expression 2];
```

- where storage-class refers to the storage class of the array, data-type is the data type, array is the array name, and expression 1 is a positive-valued integer expression which indicates the number of rows in the array while expression 2 indicates the number of columns in each row.

Defining Arrays – 2D- Examples

- Typical two-dimensional array definition is shown below:

```
int table[4][4];
```

- The statement states that table is a 16-element integer array.
  - **Elements of two – dimensional array** are stored row-wise, i.e., in the contiguous block of memory, first element of first row are stored, then elements of second row, then elements of third row, and so on.

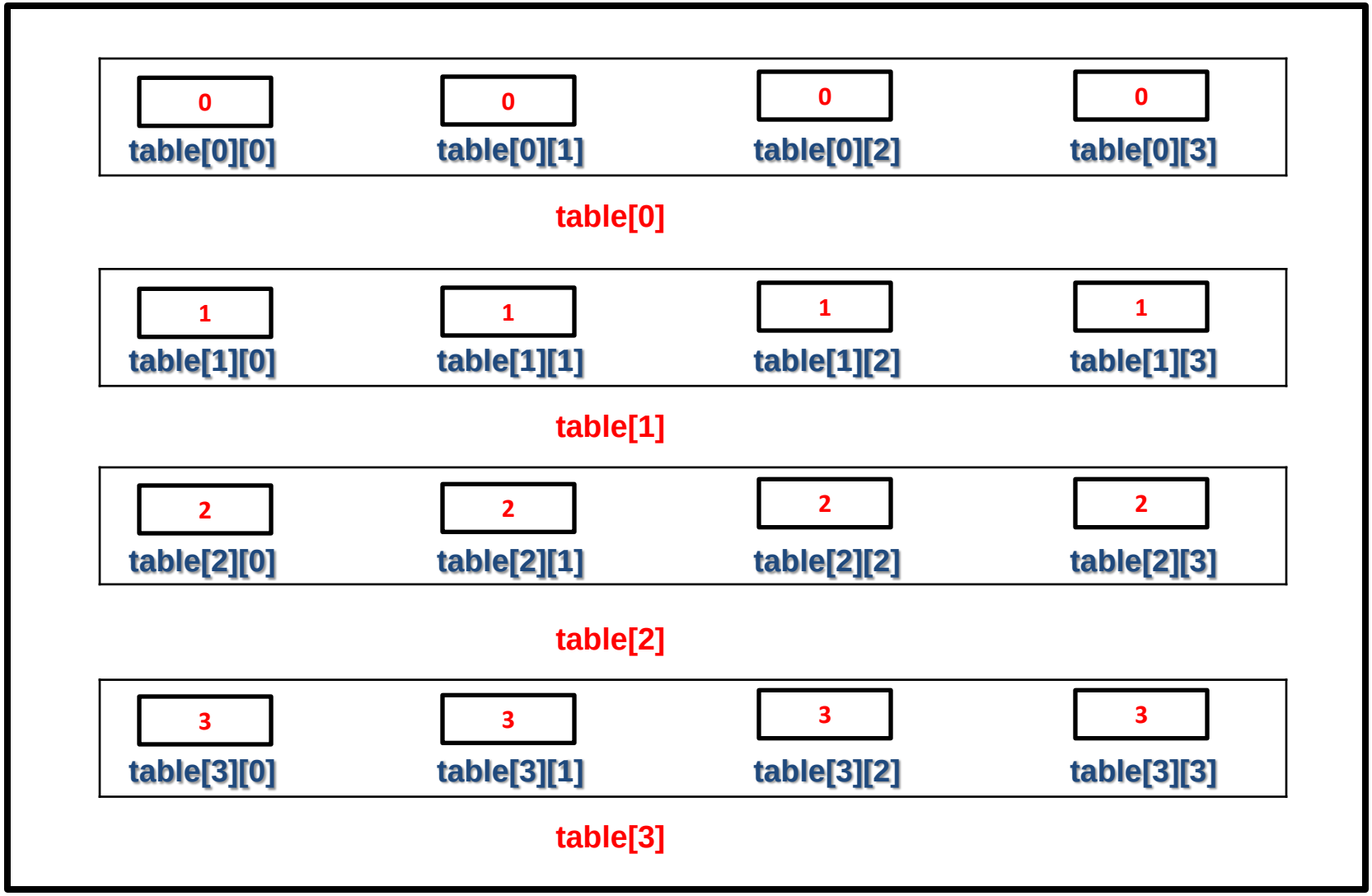
Initializing Arrays – 2D- Examples

- Shown below is an array definition that include the assignment of initial values:

```
int table[4][4] = {
                    0, 0, 0, 0, 1, 1, 1, 1,
                    2, 2, 2, 2, 3, 3, 3, 3
                  };
```

- It will initialize all the elements of the first row with value 0, elements of second row with 1, elements of third row with 2 and elements of the fourth row with value 3.

Initializing Arrays – 2D- Examples



table

Array of arrays

Initializing Arrays – 2D- Examples

- Shown below is an array definition that include the assignment of initial values (**with the help of nest braces**):

```
int table[4][4] = {
                    { 0, 0, 0, 0 },
                    { 1, 1, 1, 1 },
                    { 2, 2, 2, 2 },
                    { 3, 3, 3, 3},
                  };
```

- *Each row is initialized as a **one-dimensional array** of four elements enclosed in braces.*

- It will initialize all the elements of the first row with value 0, elements of second row with 1, elements of third row with 2 and elements of the fourth row with value 3.

Initializing Arrays – 2D- Examples

- Shown below is an array definition that include the assignment of initial values (**with the help of nest braces**):

```
int table[ ][4] = {
                    { 0, 0, 0, 0 },
                    { 1, 1, 1, 1 },
                    { 2, 2, 2, 2 },
                    { 3, 3, 3, 3},
                  };
```

- *The first dimension can be omitted if the array is* ***completely initialized****.*

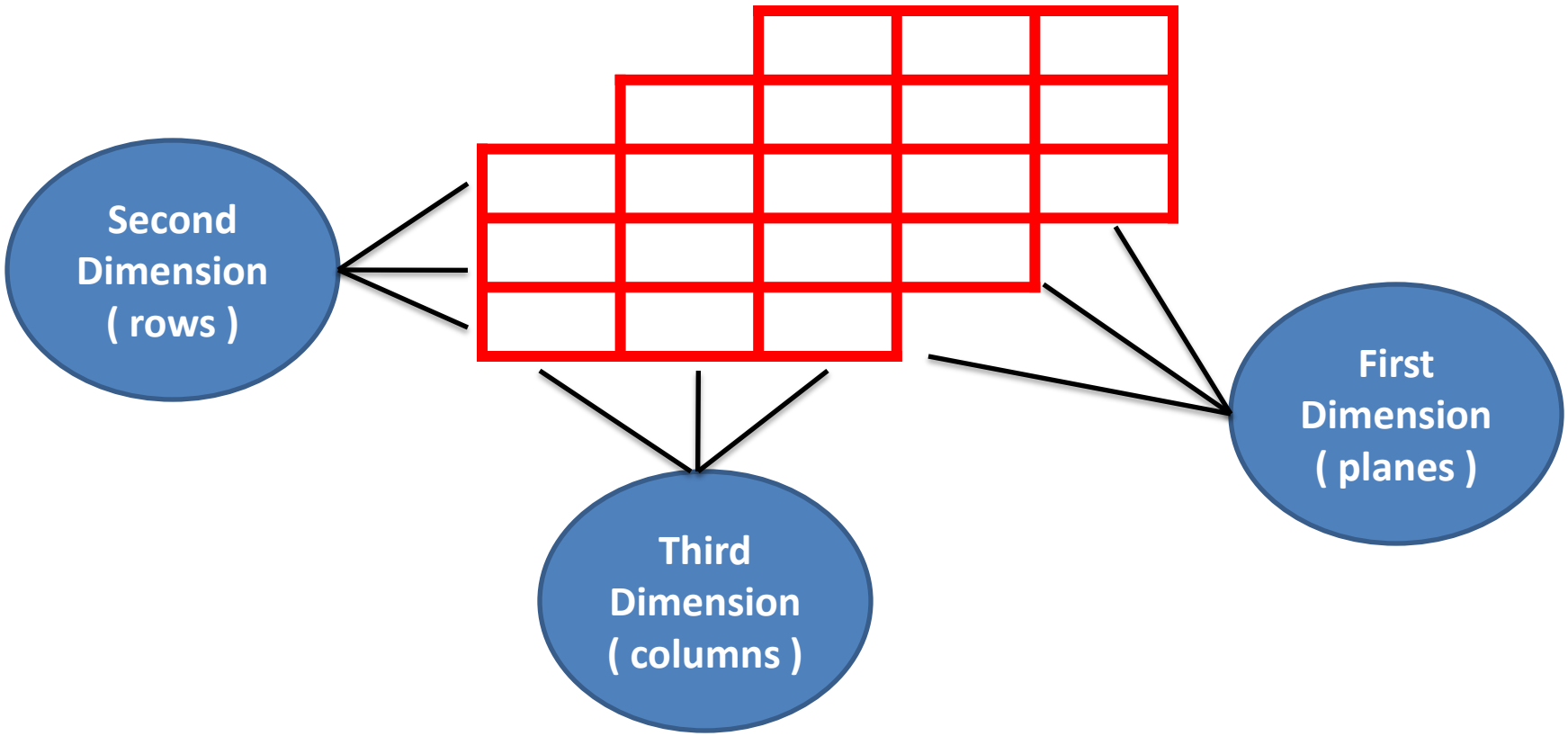
- It will initialize all the elements of the first row with value 0, elements of second row with 1, elements of third row with 2 and elements of the fourth row with value 3.

Defining Arrays – Multi-dimensional

- Multi-dimensional arrays can have three, four or more dimensions.
- The terminology used to describe the three-dimensional array is planes, rows and *columns*.
- The first dimension is called plane, which consists of rows and columns.

Defining Arrays – 3D

- Following Figure shows a three dimensional array (3×3×3):



Defining Arrays – 3D

- Although three-dimensional array is exactly what is shown in previous Figure, C language looks at it in a different way.
- *It takes a three-dimensional array to be an array of two-dimensional arrays*. *It considers the two-dimensional array to be an array of one –dimensional arrays*.
- In other words, a three-dimensional array in C language is an array of arrays of arrays. This concept also holds true for arrays of more than three dimensions.

Defining Arrays – 3D

- In general terms, a three-dimensional array definition may be expressed as:

```
storage-class    data-type    array [ expression 1 ] [ expression 2] [ expression 3];
```

- where storage-class refers to the storage class of the array, data-type is the data type, array is the array name, and expression 1 is a positive-valued integer expression which indicates the number of planes while expression 2 indicates the number of rows in an array and expression 3 indicates the columns in each row.

Defining Arrays – 3D- Examples

- Typical two-dimensional array definition is shown below:

```
int table[2][3][4];
```

- The statement states that table is a 24-element integer array organized as having:
  - **2 Planes or Pages (with plane index: 0, 1)**
  - **3 Rows in each plane (with row index: 0, 1, 2)**
  - **4 Columns in each row (with column index: 0, 1, 2, 3)**
  - **Elements of two – dimensional array** are stored row-wise, i.e., in the contiguous block of memory, first element of first row are stored, then elements of second row, then elements of third row, and so on.

Initializing Arrays – 3D- Examples

- Shown below is an array definition that include the assignment of initial values:

```
int table[2][3][4] = {
                        {                       /* PLANE 0 */
                          { 0, 0, 0, 0 },                   /* ROW 0 */
                          { 1, 1, 1, 1 },                  /* ROW 1 */
                          { 2, 2, 2, 2 }                  /* ROW 2 */
                        },
                        {                     /* PLANE 1 */
                          { 3, 3, 3, 3 },                 /* ROW 0 */
                          { 4, 4, 4, 4 },                /* ROW 1 */
                          { 5, 5, 5, 5 }                /* ROW 2 */
                        }

                    };
```

Practice Questions

- Program to compute the sum of positive values and the negative values separately. All the values are stored in an array.

- Program to split the values of the array A[] = {12.34, 25.04, 1295.50, 45.33, 1.76} into two arrays as:
  - R[] = {12, 25, 1295, 45, 1}
  - P[] = {34, 4, 50, 33, 76}

- Write a program to print the reverse of every element of the array.

- Program to print the sum of elements of every row of the array in its respective final column.

Practice Questions

- Consider an array initialized as:
  - int z[3][2][4] = {9,8,7,6,5,4,3,2,1,0,9,8,7,6,5,4,3,2,1,0,8,6,4,2};
- Draw the logical memory map for the above array.
- Determine the value represented by
  - z[2][0][1]
  - z[0][1][0]
  - z[1][1][2]
  - z[0][0][3]
  - z[1][0][1]
- Also, express the array index (in form of z[?][?][?]) where the stored data element is 6.