Arrays MultiDimensional

- The types of arrays that you have been exposed to so far are all linear arrays.
  - ➔ A single dimension
- The C language allows arrays of any dimension to be defined.
  - ➔ Two dimensional arrays are the most common.
- You can visualize a two-dimensional array as a rectangular arrangement like rows and columns in a spreadsheet.
- One of the most natural applications for a two-dimensional array arises in the case of a matrix.
- Two-dimensional arrays are declared the same way that one-dimensional arrays are

```
int matrix[4][5];
```

- Declares the array matrix to be a two-dimensional array consisting of 4 rows and 5 columns, for a total of 20 elements.
  - ➔ Note how each dimension is between its own pair of square brackets.

Initializing a two dimensional array

- Two-dimensional arrays can be initialized in the same manner of a one-dimensional array.
- When listing elements for initialization, the values are listed by row
  - ➔ The difference is that you put the initial values for each row between braces, {}, and then enclose all the rows between braces:

```
int numbers[3][4] =     {
                {10,20,30,40},          //values for first row
                {15,25,35,45},          //values for second row
                {47,48,49,50}           //values for third row
                        };
```

- Commas are required after each brace that closes off a row, except in the case of the final row.
- The use of the inner pairs of braces is actually optional, but, should be used for readability.

Initializing a 2d array

- As with one-dimensional arrays, it is not required that the entire array be initialized.

```
int matrix[4][5] =      {
                {10,5,-3},
                {9,0,0},
                {32,20,1},
                {0,0,8},
```

};

- Only initializes the first three elements of each row of the matrix to the indicated values
  - ➔ Remaining values are set to 0.
  - ➔ In this case, the inner pair of braces are required to force the correct initialization.

Designated initializers.

- Subscripts can also be used in the initialization list, in a like manner to single-dimsional arrays

int matrix[4][3]= { [0][0] = 1, [1][1] = 5, [2][2] = 9 };

- Initializes the three indicated elements of matrix to the specified values .
  - ➔ Unspecified elements are set to zero.
  - ➔ Each set of values that initialize the elements in a row is between braces.
  - ➔ The entire initialization goes between another pair of braces.
  - ➔ The values for a row are separated by commas
  - ➔ Each set of row values is separated from the next set by a comma.

Other dimensions

- Everything mentioned so far about two-dimensional arrays can be generalized to three-dimensional arrays and further/
- You can declare a three-dimensional array this way:

int box [10][20][30];

- You can visualize a one-dimensional array as a row of data.
- You can visualize a two-dimensional array as a table of data, matrix, or a spreadsheet
- You can visualize a three-dimensional array as a stack of data tables.
- Typically, you would use the three nested loops to process a three-dimensional array, four nested loops to process a four-dimensional array, and so on.

Initializing an array of more than 2 dimensions.

- For arrays of three or more dimensions, the process of initialization is extended.
- A three-dimensional array will have three levels of nested braces with the inner level containing sets of initializing values for a row.

```
int numbers[2][3][4] = {
                        {                   // First block of 3 rows
                          { 10, 20, 30, 40 },
                          { 15, 25, 35, 45 },
                          { 47, 48, 49, 50 }
                        },
                        {                   // Second block of 3 rows
                          { 10, 20, 30, 40 },
                          { 15, 25, 35, 45 },
                          { 47, 48, 49, 50 }
                        }
                      };
```

Processing elements in a dimensional array

- You need a nested loop to process all the elements in a multidimensional array.
  - ➔ The level of nesting will be the number of array dimensions.
  - ➔ Each loop iterates over one array dimension

```
int sum = 0;
for(int i = 0 ; i < 2 ; ++i)
{
  for(int j = 0 ; j < 3 ; ++j)
  {
    for(int k = 0 ; k < 4 ; ++k)
    {
      sum += numbers[i][j][k];
    }
```