Initializing | Coursera

coursera.org/learn/pointers-arrays-recursion/supplement/ks9yY/initializing

Initializing

Initializing

We can initialize a multidimensional array in the same line that we declare it by using a braced initializer, as we can for a one dimensional array. In the case of a multidimensional array, we should remember that each element of the array is itself an array, and write a braced initializer for it:

```
double myMatrix[4][3] = { {1.0, 2.5, 3.2},    //elements of myMatrix[0]
                          {7.9, 1.2, 9.9},    //elements of myMatrix[1]
                          {8.8, 3.4, 0.0},    //elements of myMatrix[2]
                          {4.5, 9.2, 1.6} };  //elements of myMatrix[3]
```

When we initialize an array in this fashion we *can* leave off the first dimension, as the compiler can determine how many elements there are from the initializer:

```
//also legal: removed the 4 from the []
double myMatrix[][3] = { {1.0, 2.5, 3.2},    //elements of myMatrix[0]
                         {7.9, 1.2, 9.9},    //elements of myMatrix[1]
```

```
                        {8.8, 3.4, 0.0},    //elements of myMatrix[2]
                        {4.5, 9.2, 1.6} };  //elements of myMatrix[3]
```

You may not elide the second dimension's size specification, even when you provide a complete initializer. You may also elide the first dimension's size when you are declaring a parameter for a function, but may not elide any other dimension's size.

A multidimensional array is not limited to two dimensions. For more dimensions, you can write additional [ ]s specifying the size of each additional dimension:

```
int x[4][2][7];  //x is a 3D array, with 4 elements, each of which is
                 //an array with 2 elements
                 // (whose elements are 7-element arrays of ints)
char s[88][99][122][44]; //s is a 4D array of chars: 88 x 99 x 122 x 44.
```

All of the same rules apply to these arrays with more dimensions. If we write *x*[*1*], we get a pointer to the 2-D array which is the 1st element of *x*. If we write *x*[*1*][*1*], we get a pointer to the 1-D array of ints which is the 1st element of that array, and if we write *x*[1][*1*][*4*], we get int which is the 4th element of that array. We can also initialize these arrays with braced initializers if we want to (although writing the initializer for a large array with many dimensions will likely take a significant amount of time).

✓

Completed