ECE 131 – Programming Fundamentals

Dr. Daryl O. Lee

University of New Mexico

Two-dimensional Arrays

The C syntax for declaring a two-dimeniensional array:

$$type\ identifier[row][col] <= \{initialization\ string\} >;$$

where *type* is any valid C data type, either built-in or user-defined, *identifier* is the name of the array, which can be any valid C identifier, *row* and *col* are C expressions that specify how many rows and columns, respectively, will be in the array, and an optional *initialization string*, enclosed in curly braces, can be used to store initial values in the array.

Ex: `int a[2][3] = {{3,1,5},{2,2,4}};`

Note 1: The total number of elements in a two-dimensional array is ($row \cdot col$), i.e., 6 in this example.
Note 2: The following is invalid C syntax, and will yield a compiler error: `int a[2,3]; // error!`

THE UNIVERSITY of NEW MEXICO

Two-dimensional Arrays

- You can think of a two-dimensional array as a table.
  Ex: `int a[2][3] = {{3,1,5},{2,2,4}};`

	0	1	2
0	3	1	5
1	2	2	4

- Once again, the row and column numbering is zero-based.
- To index into the array, you use its row and column index.
  Ex: `a[1][0] = 6;`

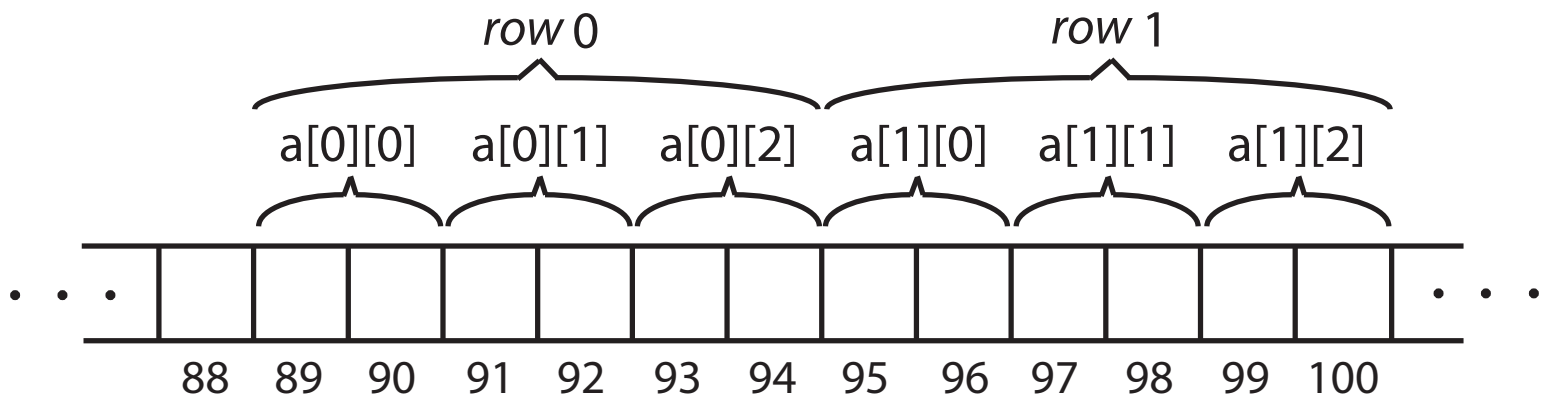
	0	1	2
0	3	1	5
1	6	2	4

THE UNIVERSITY of NEW MEXICO

Two-dimensional Array Storage

- Just like one-dimensional arrays, all of the elements in a multidimensional array are stored in contiguous memory locations.
- More specifically, a two-dimensional array is stored in contiguous locations in memory in row-column order.

Ex:



Note: The column dimension varies most rapidly when considering the array elements in storage order.

THE UNIVERSITY of NEW MEXICO

Two-dimensional Array Storage

- If you remove the inner set of curly braces from the declaration:

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

  i.e.,

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

  the C compiler will simply initialize the array in storage order. Thus, the two declarations above are equivalent.

- This is also a valid array declaration:

  ```
  int a[2][3] = {{3,1},{2,2}};
  ```

  It does not initialize the last element in each of the two rows in the array.

- This is *not* the same as the previous declaration:

  ```
  int a[2][3] = {3,1, ,2,2, }; // error!
  ```

  In fact, it will produce a compiler error.

THE UNIVERSITY of NEW MEXICO

Multidimensional Arrays

The C syntax for declaring a multidimensional array is a straightforward extension of what we've already seen. I.e., to declare an n-dimensional array, you need to provide a value for the size of each of the n dimensions:

$$type\ identifier[size_1][size_2]\cdots[size_n] <= \{initialization\ string\} >;$$

Ex: `int b[3][4][2][4];`

- Array `b` has four dimensions, and a total of $3 \cdot 4 \cdot 2 \cdot 4 = 96$ `int` elements.
- The statement

  ```
  b[0][0][0][1] = 5;
  ```

  assigns the value 5 to the the second element (in storage order) of the array.

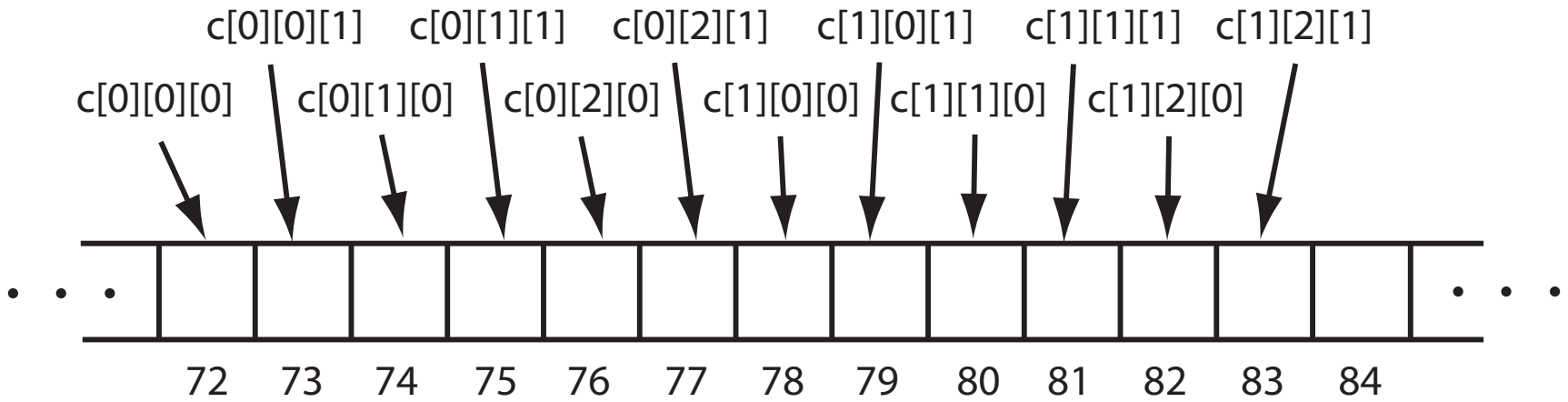
THE UNIVERSITY of NEW MEXICO

Multidimensional Arrays

When considering a multidimensional array's elements in storage order, the rightmost index varies most rapidly, and the leftmost index varies least rapidly.

Ex: `char c[2][3][2];`

Will lead to the following storage order in memory:



THE UNIVERSITY of NEW MEXICO

Multidimensional Arrays

In order to determine how to add curly braces for the initialization string, it's easier to think in the other direction, i.e., right-to-left.

```
char c[2] = {      ,      };
char c[2][3] = {{   ,   ,   }, {   ,   ,   }};
char c[2][3][2] = {{{ , },{ , },{ , }},{{ , },{ , },{ , }}};
```

Ex. The following declaration:

```
char c[2][3][2]={{{0,3},{1,3}},{{1,4},{0,2}},{{2,5},{1,1}}};
```

will produce: