Lecture 07 Notes

Derek Harter

CSci 515 Spring 2015 *<2015-02-18 Wed>*

1 First Session (11 - 11:40)

1.1 Two-Dimensional Arrays

- Two-dimensional arrays represent **TABLES** of values
- Tables are a very fundamental data structure/concept. Represent the basic type of a relational database, for example.
- Information arranged in rows and columns (we have seen this before in our lab where we looked at DSV separated data).
- By convention, the first identifier for a 2-D array specifies the table's row, and the second identifier specifies the column.
- This makes sense, as the row is the **record** or **trial** of a table e.g. it is the set of related information.
- The column is a particular attribute or piece of information about a record or trial.
- So, to initialize a 2-D array:

```
const int NUM_RECORDINGS = 5; // number of rows in table
const int NUM_DIMENSIONS = 3; // number of columns in table

// Each recording records a 3-dimensional position of a particle in
// our experiment
float experimentPositions[NUM_RECORDINGS][NUM_DIMENSIONS];
```

- For 1-D arrays, the size of the array was not required by compiler when passing array (by reference) to a function. Though we always specified the size of the array as the next parameter for our functions.

- For 2-D arrays, the number of rows (the first dimension) is still not required by compiler, though again we will almost always pass this information in to the function because it is needed to process the 2-D array.
- However, the compiler needs the number of columns to be specified, because reasons.
  - reasons: arrays are stored using row major ordering, thus we need to know the number of columns in order to correctly calculate item position.
- 2-D arrays require a nested loop (2 loops) to process/init
- Function to initialize our 2-D array

```
void initExperimentArray(float values[][NUM_DIMENSIONS], int size)
{
  // iterate over all of the rows/records of the array
  for (int record = 0; record < size; record++)
  {
    for (int dim = 0; dim < NUM_DIMENSIONS; dim++)
    {
      values[record][dim] = 0.0;
    }
  }
}
```

1.2 Two-Dimensional Array Layout in Memory

- Show example of 2-D array in memory using debugger.

2 Second Session (11:45 - 12:30)

2.1 Example of processing a 2-D array

- Initialize 2-D array
- Read 2-D array from file of DSV values
- Display 2-D array as a table
- Process 2-D array

2.2 Three and higher-dimensional arrays

- Allocate a 3-D array.
- Initialize a 3-D array using 3 nested loops.

3 Third Session (12:40 - 1:40)

3.1 Array operations that modify size of array

- Shifting elements in an array
- Add element to an array
- Remove element from an array
- Swap element with first element