Arrays Episode IV: A New Array

CS 211

Adam Sweeney

October 13, 2017

Wichita State University, EECS

Introduction

- This section is short, and serves more as an FYI
- How would we represent something like a matrix with an array?

Agenda

- Section 7.4
- Declaring and using
- Passing to Functions

Declaring & Using Multi-Dimensional Arrays

Declaring & Using Multi-Dimenstional Arrays

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

- Declaring a multi-dimensional array is not so different from declaring a regular array
- We add an extra set of [ ] to indicate an added dimension, and sizes are required for both dimensions
- We are not limited to two-dimensional arrays, but beyond three, it gets very difficult to visualize

Accessing a Specific Element

- In a 2D array, the first dimension can be thought of as the row number
- The second dimension is the column
- To access a single element, we must specify a row and a column
  - In other words, we have to fully specify where in the array the element is

```
// Given the following array
int matrix[5][5];

// The following call
int x = matrix[3][1];
// Stores the value in the fourth row, second column
// of the array in the variable x
```

Passing an Element to a Function

- This works exactly the same as it does for a 1D array
- We simply have to remember to give the complete location (specify where the element is located in every dimension) of the element

Passing a Multi-Dimensional Array to a Function

Passing a Multi-Dimensional Array to a Function

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

- Given the above matrix, the following function prototype:

```
void foo(int arr[][5], int size);
```

- This is how we would pass our matrix to the function

```
foo(matrix, 5);
```

- Note the similarities, the prototype's first dimension size is blank, as we would expect
- The new syntax comes from the fact that the second dimension does have a size listed in the [ ]

But Why, Though?

- Let us consider how an array is stored memory
- We should imagine our computer's main memory as a long, straight line of bytes
- The only way to represent anything is by occupying a chunk of the line
- We have seen that arrays occupy however many bytes they need to hold all of their elements
- So, this means that instead of a 2D array, we need to think of it as an array of arrays
- By requiring the size of all dimensions beyond the first, we are making sure compiler knows how much memory to reserve

An Array of Arrays

- The idea here is that we know how big our basic data types are
- A `char` is one byte, an `int` is typically 4 bytes, etc.
- We don't automatically know how big an array is, unless we know how many elements are in the array
- Because a 2D array is really an "array of arrays", we have to specify the size of the item that our main array is storing