Arrays
Feb. 26, Ch 7

[“Hip” , “Hip”]

Hip Hip Array
- arrays in functions

```c
double arr[5];
foo(arr);
```

- 2D arrays

```c
int box[3][4];
// 3 rows, 4 columns
```
Array - array passing

Arrays are references (memory addresses)

This means we can pass the reference as an argument in a method

Then the method can see the whole array, but it won't know the size

(See from last time: maxPassArray.cpp)
Array - array passing

But wait! This means the function can change the data since we share the memory address

(See: reverse.cpp)
Array - array passing

If we want to prevent a function from modifying an array, we can use `const` in the function header:

```c
void reverse(const int word[]);
```

This also means any function called inside reverse must also use `const` on this array.

(See: reverseFail.cpp)
Array - returning arrays

However, we do not know how to return arrays from functions (yet)

```c
int[] foo(){
    int x[] = {1,2};
    return x;
}
```

// x dies here, what are you returning?

For now, you will have to pass in an array to be changed, much like call-by-reference
Let's practice arrays by sorting!

(See: sort.cpp)
Sort

Let's practice arrays by sorting!

Plan of attack:

1. Make a new array
2. Find minimum element in original array and copy into new array
3. Replace minimum element in original array with the maximum element
4. Repeat 2 to 3 until done

(See: sort.cpp)
Multidimensional Arrays

So far we have dealt with simple (one dimensional) arrays

We have represented this as all the data being stored in a line

(See: lineWorld.cpp)
Multidimensional Arrays

```java
int foo[][] = new int[3][5];
```

foo's length = 3
(number of rows)

foo[0]'s length = 5
(number of columns in row 0)
Multidimensional Arrays

If we think of a couple simple (one dimensional) arrays on top of each other...

One array for numbers 1-10

One array for numbers 71-80

(See: gridWorld.cpp)
Multidimensional Arrays

Recreate:

(See: oneToAHundred.cpp)