Arrays & functions

Ch 7

Announcements

Quiz scores on gradescope

Test next Wednesday (whole class period)

Arrays - looping

As arrays store multiple elements, we very often loop over those element

There is a special loop that goes over all elements (for each):

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

for(int a : x)
{
```

x is an array

a has the value of x[i] for each i

(See: forEach.cpp)

Partially filled arrays

Arrays are annoying since you cannot change their size

You can get around this by making the array much larger than you need

If you do this you need to keep track of how much of the array you are actually using

(See: partiallyFilled.cpp)

Array - element passing

Each element of an array is the same as an object of that type

For example: `int[] x = {1, 2};`
x[0] is an int, and we can use it identical as if we said: `int x0 = 1;`

(See: maxPassInt.cpp)

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: 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:

```
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)

```
int[] foo(){
    int x[] = {1,2};
    return x;
} // x dies here, what are you returning?
```

syntax error

For now, you will have to pass in an array to be changed, much like call-by-reference