Arrays

Oct. 20, Ch 7-8.2

Why science teachers are not asked to monitor recess.
Highlights

- make an array (store multiple objects)

```cpp
int x[5]; // 5 ints
```

- access array

```cpp
x[0] = 2;
cout << x[0];
```
Arrays are convenient ways to store similar data types.

Much like how we have been thinking of strings as being multiple chars.

Arrays are indexed starting from 0, so index 0 is the first element, index 1 is the second element...
Arrays

When you make an array, the computer reserves space in memory for the size.

The array variable is then just a reference to the first element's memory location.

The computer simply converts the index into an offset from this initial location (see arrayAddress.cpp)
Memory

Memory:

CAUTION OFF LIMITS  CAUTION OFF LIMITS

Code:
Memory (declaration)

Memory:

#0 (int) x

Code:

int x;
Memory (declaration)

Memory: y is the address of y[0]

#0 (int) x #1(int)y[0] #1(int)y[1] #1(int)y[2]

Code:

```
int x;
int y[3];
```
Arrays - declare-initialization

When making an array, you need both a type and a length.

The format for making an array is below:

```
int x[5];  // 5 ints
```

- `int`: variable type
- `x`: variable name
- `[5]`: length of the array

Type, `[]` means array length of array

variable name
Arrays - elements

To access an element of an array, use the variable name followed by the index in [ ]

\[ x[1] = 2; \]

(See: simpleArray.cpp)
Arrays - manual initialization

Arrays can be initialized by the following:

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

If you access outside of your array you will either crash or get a random value

You can also use a constant variable to set the size:

```c
const int size = 8;
int x[size];
```

(See: average.cpp)
Arrays - looping

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

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

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

for (int a : x) {
    // a has the value of x[i] for each i
}
```

(See: forEach.cpp)
Partially filled arrays

POINTS OF VIEW

- **OPTIMIST**
  - "The glass is half-full."

- **PESSIMIST**
  - "The glass is half-empty."

- **REALIST**
  - "Yep. That's a glass, alright."

- **IDEALIST**
  - "One day, cold-fusion from a glass of water will provide unlimited energy and end war."

- **CAPITALIST**
  - "If I bottled this and gave it a New Agey sounding name, I could make a fortune."

- **COMMUNIST**
  - "This drink belongs to every single one of us in equal measure."

- **CONSPIRACIST**
  - "The government is fluoridating the water for mind-control purposes."

- **SEXIST**
  - "This glass isn't gonna refill itself, honeybun..."

- **NIHILIST**
  - "The glass does not exist, and neither do I."

- **OPPORTUNIST**
  - "There's a funny t-shirt in here somewhere."
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

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

```plaintext
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
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 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

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)
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)