TODO: record lecture

Week02

- arrays
- array examples
- swap
- array examples that use swaps

WARMUP
Guess how many arrays we used to make this:
https://youtu.be/dadIdXQMBi4

NOTE: an **array** is a fixed-length sequence of elements, all the same type

how many arrays?

- ARRAYS (not counting tiny arrays of 3 double's, which i use to represent points)
- NOTE: Counts are very approximate
- ==========
- strings for loading files >5
- huge arrays for drawing random stuff (the floor, etc) 2
- random stuff in my cow.cpp library (keyboard state, OpenGL objects, etc.) >20
- random global variables 1
- building the simulation >6
- simulation state 3
- simulation itself >20
- dragon head mesh 7
- dragon body mesh 7
- dragon mesh 7
- sphere mesh 3
- cylinder mesh 3
- motor configuration 3
- arrays that just move data back and forth between C++ <-> unity 8
- inverse kinematics >15
- physics >10
- ---------
- definitely at least 100; probably less than 1000

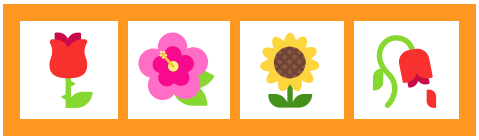
arrays

metaphor

an array is like a very organized person's flower planter

- one flower per square
- the planter can't change size

(it is made of artisinal woods or something)



mental model of an array

mental model of an array

- a fixed number of equally-sized boxes, all right next to each other
- the array itself lives "in memory"

1	5	3	2	7

```
int[] foo = new int[5];
foo[0] = 1;
foo[1] = 5;
foo[2] = 3;
foo[3] = 2;
foo[4] = 7;
```

- the array itself lives "**in memory**"

1	5	3	2	7

learn about **memory** and become a real CS pro

a couple important questions to ask yourself:
https://www.youtube.com/watch?v=BNtcWpY4YLY

arrays

arrays (1/2)

- an **array** is a fixed-length sequence of elements, all of the same type
  - "an array of 4372 `double`'s"
  - "an array of 1 `int`'s"
  - "an array of 64 `Student`'s"

arrays (2/2)

- we will often write an array using curly braces
  - `{ 7, 7, 9 }` is an array containing `7`, `7`, and `9`
- optionally, you can include a comma after the last element
  - `{ 7, 7, 9, }`

- ☠ **even though sets from math also use curly braces, Java arrays have nothing to do with sets; in a set, all elements must be unique; in an array, elements do NOT have to be unique**

array operations

array operations

creating an array (1/2)

- you can create an array by specifying its **length** (the number of elements); if you do, the array is **zero-initialized** (all elements are initially set to zero)
  - `int[] A = new int[8]; // { 0, 0, 0, 0, 0, 0, 0, 0 }`
  - `double[] B = new double[1]; // { 0.0 }`
  - `boolean[] C = new boolean[2]; // { false, false }`
  - `char[] D = new char[4]; // { '\0', '\0', '\0', '\0' }`
  - `String[] E = new String[3]; // { null, null, null }`

  - [visualize in Eclipse]

creating an array (2/2)

- you can also create an array by specifying its elements; if you do, the array's length is the number of elements you specified
  - `int[] array = { 7, 7, 9 }; // int[3] is implied`
    - `NOT OKAY to do later: array = { 4, 5, 6 };`
    - `OKAY to do later:`
    - `array[0] = 4;`
    - `array[1] = 5;`
    - `array[2] = 6;`
  - `boolean[] array = { true };`
  - `String[] array = { "hello", "world" };`

getting an array's length

- after creating an array, you can get (but not set) its length

```
int[] array = { 7, 7, 9 };
PRINT(array.length); // 3
```

```
int[] array = new int[8];
PRINT(array.length); // 8
```

```
Error: cannot assign a value to final variable length

array.length = 42;
```

getting the value of an element of an array

- you can **get** the value of an element of an array using the square brackets and the index of the element
  - this is also called "accessing the array"

```
int[] array = { 3, 4, 5 };
int foo = array[0]; // 3
```

```
int[] array = { 3, 4, 5 };
int foo = array[42];

java.lang.ArrayIndexOutOfBoundsException: 42
```

setting the value of an element of an array

- you can **set** the value of an element of an array using the square brackets and the index of the element

```
int[] array = { 7, 7, 9 };
array[1] = 8;
// { 7, 8, 9 }
```

```
int[] array = { 7, 7, 9 };
array[-1] = 1000;

java.lang.ArrayIndexOutOfBoundsException: -1
```

☠️ printing the elements of an array

- ☠️ **in Java, you don't simply call** `System.out.println(array)`
  - **instead, you call** `System.out.println(Arrays.toString(array));`
    - **note: this prints with square brackets instead of curly brackets**
      - 🤷
- **or just call** `PRINT(array);`
  - 🐄💔✨
  - (note: also uses square brackets)

iterating over an array

iterating over an array

- a for loop can be used to iterate over an array

```
for (int i = 0; i < array.length; ++i) {
    array[i] = ...;
}
```

[example in Eclipse]

array examples

example: creating an array of the first 100 non-negative integers

example: creating an array of the first 100 non-negative integers [0, 1, ..., 99]

```
class Main extends Cow {
   public static void main(String[] arguments) {
      int[] array = new int[100];

      for (int i = 0; i < array.length; ++i) {
         array[i] = i;
      }

      PRINT(array);
   }
}
```

example: array copy

example: array copy

```
class Main extends Cow {
    public static void main(String[] arguments) {
        int[] source = { 3, 4, 5 };

        int[] destination = new int[source.length];
        for (int i = 0; i < source.length; ++i) {
            destination[i] = source[i];
        }

        PRINT(source);
        PRINT(destination);
    }
}
```

example: bad very bad broken array copy

example: bad very bad broken array copy

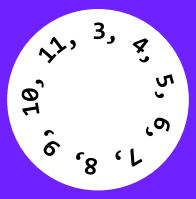
```
import java.util.*;

class Main {
    public static void main(String[] arguments) {
        int[] source = { 3, 4, 5 };

        int[] destination = source;

        source[0] = 7;
        System.out.println(Arrays.toString(source));
        System.out.println(Arrays.toString(destination));
    }
}
```

example: circular array



example: circular array

```
import java.util.*;

class Main {
    public static void main(String[] arguments) {
        int[] array = new int[5];
        int indexToWriteIntoNext = 0;
        while (true) {
            array[indexToWriteIntoNext] = getIntFromUser();
            indexToWriteIntoNext = (indexToWriteIntoNext + 1) % array.length;
            System.out.println(Arrays.toString(array));
        }
    }

    static int getIntFromUser() {
        Scanner scanner = new Scanner(System.in);
        while (!scanner.hasNextInt()) { scanner.nextLine(); }
        return scanner.nextInt();
    }
}
```

example: finding the index (and value) of an array's maximum element

example: finding the index (and value) of an array's maximum element

```
import java.util.*;

class Main {
    public static void main(String[] arguments) {
        double[] array = { 1.0, 3.0, -42.0, 1000.0, 99.0 };

        int indexOfMaximumElement = -1;
        double valueOfMaximumElement = Double.NEGATIVE_INFINITY;
        for (int i = 0; i < array.length; ++i) {
            if (array[i] > valueOfMaximumElement) {
                indexOfMaximumElement = i;
                valueOfMaximumElement = array[i];
            }
        }

        System.out.println("array[" + indexOfMaximumElement + "] = " + valueOfMaximumElement);
    }
}
```

multi-dimensional arrays

multi-dimensional arrays

multi-dimensional arrays (arrays of arrays of ...)

- multi-dimensional arrays are sometimes really handy

```
int[][] array = { { 3, 4 }, { 5, 6 }, { 7, 8 } };
System.out.println(Arrays.deepToString(array));
// [[3, 4], [5, 6], [7, 8]]
System.out.println(array[0][1]); // 4
```

```
int[][][] array = new int[2][3][4];
array[0][0][0] = 42;
System.out.println(Arrays.deepToString(array));
// [[[42, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]], [[0, 0, 0, 0],
[0, 0, 0, 0], [0, 0, 0, 0]]]
```

swap

Python swap

Python swap

```
a = 0
b = 1

a, b = b, a # Python swap

# a is now 1
# b is now 0
```

☠ BAD VERY BAD BROKEN swap

☠ BAD VERY BAD BROKEN swap

```
int a = 0;
int b = 1;

{ // BAD VERY BAD BROKEN swap
    a = b; // a <- 1
    b = a; // b <- 1
}
```

swap

swap

```
int a = 0;
int b = 1;

{ // swap
    int tmp = a; // tmp <- 0
    a = b; // a <- 1
    b = tmp; // b <- 0
}
```

array algorithms that use swaps

reversing an array

out-of-place reverse

- we can reverse an array **out-of-place** using an additional array

```
static int[] outOfPlaceReverse(int[] array) {
    int[] result = new int[array.length];
    for (int i = 0; i < array.length; ++i) {
        int j = (array.length - 1) - i;
        result[i] = array[j];
    }
    return result;
}
```

in-place reverse

- we can reverse an array **in-place** using "swaps" (no additional array)

```
static void inPlaceReverse(int[] array) {
    for (int i = 0; i < array.length / 2; ++i) {
        int j = (array.length - 1) - i;
        int tmp = array[i];
        array[i] = array[j];
        array[j] = tmp;
    }
}
```

bubble sort

bubble sort

- **bubble sort** is a simple in-place sorting algorithm that uses swaps

```
import java.util.*;
class Main {
    public static void main(String[] arguments) {
        int[] array = { 2, 4, 1, 6, 0, 3, 5 };

        boolean arrayIsSorted;
        do {
            arrayIsSorted = true;
            for (int i = 0; i < array.length - 1; ++i) {
                if (array[i] > array[i + 1]) {
                    arrayIsSorted = false;
                    // swap
                    int tmp = array[i];
                    array[i] = array[i + 1];
                    array[i + 1] = tmp;
                }
            }
        } while (!arrayIsSorted);

        System.out.println(Arrays.toString(array));
    }
}
```

big O

big O

big O (1/2)

- **big O** describes a function's "limiting behavior"
  - to find a mathematical function's big O notation...
    - 1. throw away the coefficients
    - 2. find the fastest growing term
    - 3. the function is $\mathcal{O}$(FASTEST_GROWING_TERM)
  - **e.g.,** $f(n) = 7n^2 + 100n + 4732$
    - 1. throw away coefficients to get $n^2 + n + 1$
    - 2. fastest growing term is n^2
    - 3. $f(n)$ is $\mathcal{O}(n^2)$

big O (2/2)

- what is $f(n) = 77n^7 + 2^n$ in big O notation?
  - $n^7 + 2^n$
  - 2^n is this true?
  - $\mathcal{O}(2^n)$

1. throw away the coefficients
2. find the fastest growing term
3. the function is $\mathcal{O}$(FASTEST_GROWING_TERM)

- what is $f(n) = 100$ in big O notation?
  - 1
  - 1 what does this *mean*?
  - $\mathcal{O}(1)$
- what is $f(n) = n + \log(n)$ in big O notation?
  - $n + \log(n)$
  - n is this true?
  - $\mathcal{O}(n)$

🗣️ what is the big O **`digitSum(int n)`**?

can you do better?

Caaaaaaaaaaarl

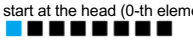
- **e.g.,** Imagine a classroom with n students. I want to figure out if any students are named Carl.
  - I need an ✨Algorithm✨ -- `boolean isAnyoneNamedCarl(Student[] students);`
  - What is the big O of the following algorithms?
    - **Algorithm 1:** Ask each student, one at a time, "Are you named Carl?"
    - **Algorithm 2:** Pass a paper around the room, and have each student write their name on it. Then take the paper, and read through it.
    - **Algorithm 3:** The students draw straws one at a time. The student who draws the short straw must leave. On their way out of the room, ask them whether their name is Carl. Repeat this procedure until the room is empty.
    - **Algorithm 4:** Play Kahoot. The winner legally changes their name to Carl.

runtime of array operations

runtime of array operations

- arrays are fast!
  - ⏱️ creating an array takes $\mathcal{O}(n)$ time, where n is the length of the array
  - ⏱️ getting the value of the i-th element of an array takes $\mathcal{O}(1)$ time
  - ⏱️ setting the value of the i-th element of an array takes $\mathcal{O}(1)$ time

accessing an array "under the hood"

- how Java does `array[4]`, step-by-step:
  - start at the head (0-th element) of the array
    
  - move over 4 slots (using an $\mathcal{O}(1)$ "add")
    
  - return the value of the element in that slot
    
- we can't actually see Java do this (it's too "low level")
  - but we *can* see C do it! (stay tuned 😀)

💻 array tutorial

💻 array tutorial

(this time with a wiki page)

💻 array tutorial

```
// 🟢 ([5, 4, 7, 0, 0, 7], 7) -> 2
// 🟢 ([5, 4, 7, 0, 0, 7], 3) -> -1
static int findFirstIndexOf(int[] array, int value);
```

```
// 🟦 ([1, 3, 3, 2, 3, 4], 3) -> [1, 2, 4]
static int[] removeAllOccurences(int[] array, int value);
```

```
// ♦️ [[1, 3], [2, 4, 5], [0, 5]] -> [0, 1, 2, 3, 4, 5, 5]
static int[] mergeArrayOfSortedArrays(int[][] arrays);
```