THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

COMP 110
Introduction to Programming

Fall 2015

Time: TR 9:30 – 10:45

Room: AR 121 (Hanes Art Center)

Jay Aikat

FB 314, *aikat@cs.unc.edu*

Previous Class

- What did we discuss?

Today

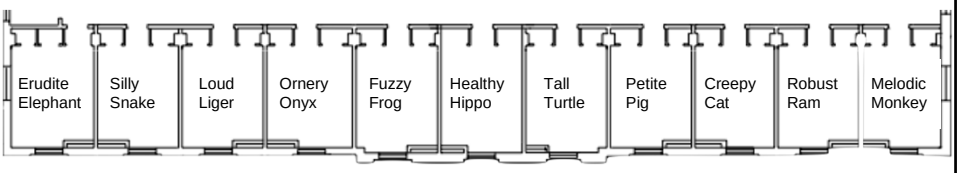
- Announcements
  - Assignment 2 : Due Friday, Oct 2 @ 11:55 PM

    http://comp110.com/assignments/the-worried-wizard
- Midterm on Thu, Oct 8
  - in class, no computers
- Study guide

  http://comp110.com/midterm-study-guide
- Arrays

Proposal: Krzyzewski Dorm

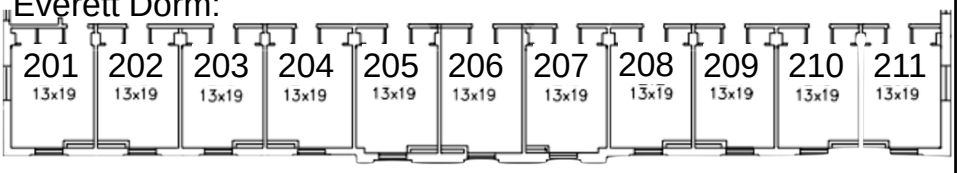
Mike Krzyzewski will donate a lot of money to UNC iff:
In Coach K Dorm, the rooms aren't numbered, they're named.



What benefits do room numbers provide?

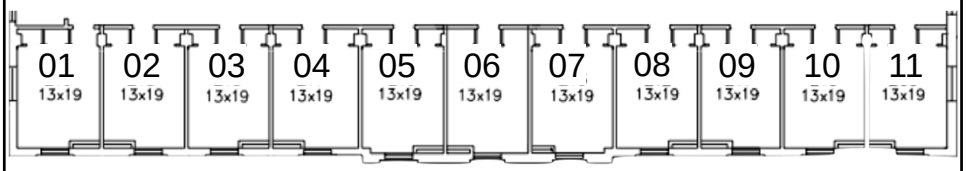
What are the benefits of a
Dorm Name + Number
Addressing Scheme?

Everett Dorm:



- Naming is hard. Numbering is easy.
  Learning and remembering names even harder.
- Locate rooms quickly and predictably.
- Easier to manage A LOT of rooms.
  "Ok, I'll prep rooms 1-150, you prep 151-300."

Arrays are like **Dorms for Data**



int	int	int	int	int	int	int	int	int	int	int
0	1	2	3	4	5	6	7	8	9	10

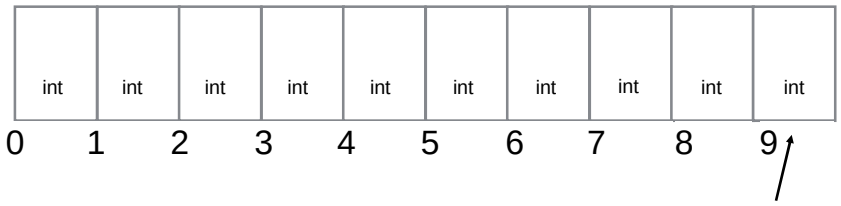
(Not drawn to scale.)

Arrays provide uniform housing for *many* values.

int	int	int	int	int	int	int	int	int	int

1. Each “room” in an array is called an **Element**
2. An element stores a single value. *No roommates!*
3. All elements in an array are of the same type
4. Arrays cannot be resized after construction

Elements are addressed by **Name and Index**



1. Notation: `arrayName[index]`, i.e. `arrayName[9]`
2. **Indexing starts at `[0]`** (not `[1]`)
   (Beware: Off-by-one bugs can be stingers 🐝.)

Questions?

Where have we seen []s?

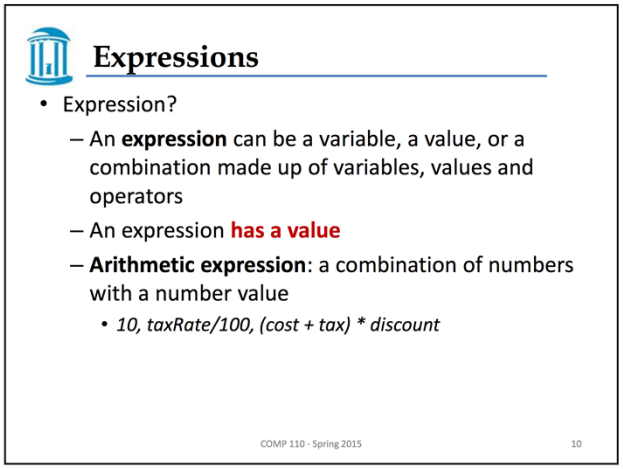
Let's demystify some magic

```
public static void main(String[] args)
```

- When you see [brackets] there are arrays
- This is an array of strings available to our program!
- What's in it?
  - Arguments passed in when we run our program
  - `> java cli.CommandLineDemo`

foo	bar	baz
0	1	2

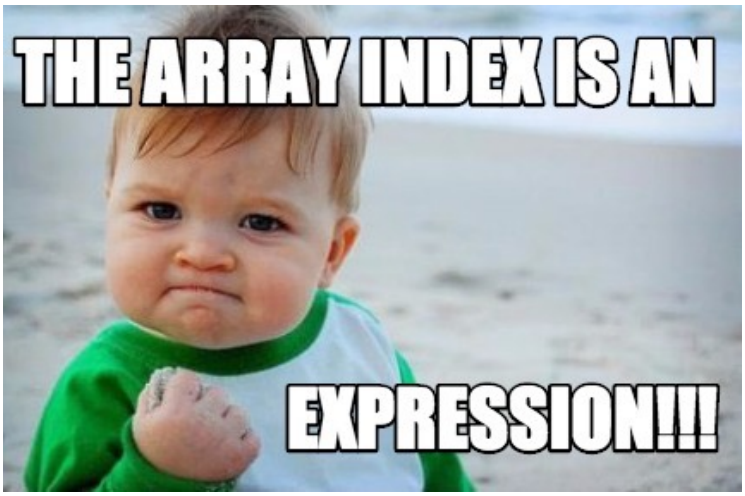
Recall Expressions:



Expressions

- Expression?
  - An **expression** can be a variable, a value, or a combination made up of variables, values and operators
  - An expression **has a value**
  - **Arithmetic expression**: a combination of numbers with a number value
    - *10, taxRate/100, (cost + tax) * discount*

COMP 110 - Spring 2015 10



This is a **major coup**.

Operation	Form	Example
Read Element	`arrayName[index]`	`scores[0]`

Accessing Elements with an Index

`index` is any integer expression.

Number
```
arrayName[0]
```

Integer
```
int i = 0;
arrayName[i]
```

Arithmetic
```
int i = 0;
arrayName[i+1]
```

Looping with an Integer
```
for(int i = 0; i<arrayName.length; i++){
  System.out.println(arrayName[i]);
}
```

Operation	Form	Example
# of Elements	`arrayName.length`	`scores.length`

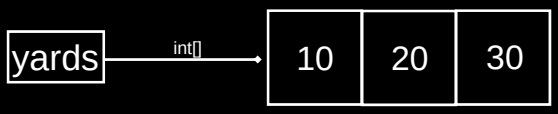
Finding # of Elements

`.length` property is number of elements in array.

Use "**< a.length"** in for loop termination test.

How do we make our own arrays?

Declare an Array	`int[] yards;`
Construct an Array	`yards = new int[3];`
Assign Elements	`yards[0] = 10;`
	`yards[1] = 20;`
	`yards[2] = 30;`



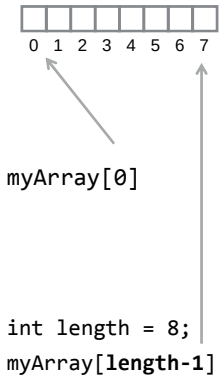
How do we average **yards**?

Java Array Operations

Operation	Form	Example
# of Elements	`arrayName.length`	`scores.length`
Read Element	`arrayName[index]`	`scores[0]`
Declare	`type[] arrayName;`	`int[] scores;`
Construct	`arrayName = new type[size];`	`scores = new int[3];`
Assign Element	`arrayName[index] = expression;`	`scores[0] = 12;`
Initialize (Just a shortcut.)	`type[] arrayName = {elements};`	`int[] scores = {12,0,1};`

Key Concepts

1)	An array is a uniform structure housing many elements.
2)	Elements are addressed via name and index.
3)	***Index is an expression!!!***



Arrays

- To think about arrays, let's think about loops first
- Why do we need loops?
  - Because we want to repeat things without writing them again and again

Average Score without Loops

- Assuming that we only need 5 basketball scores for averaging…

```
int score1 = keyboard.nextInt();
int score2 = keyboard.nextInt();
int score3 = keyboard.nextInt();
int score4 = keyboard.nextInt();
int score5 = keyboard.nextInt();

double average = (double) (score1 + score2 +
            score3 + score4 + score5) / 5.0;
```


Average Score with Loops

- Assuming that we only need 5 scores

```
for (int i = 0; i < 5; i++)
      scoreSum += keyboard.nextInt();

double average = (double) scoreSum / 5.0;
```

What if we really need to save them

- If we really need to save these scores, loop won't help you
- Think about this problem
  - Print out if a score is above/below average
  - We have to calculate average first, then decide if a score is above/below average
  - Therefore we must save all these scores, and compare them to the average in the end

Comparing All Scores and the Average

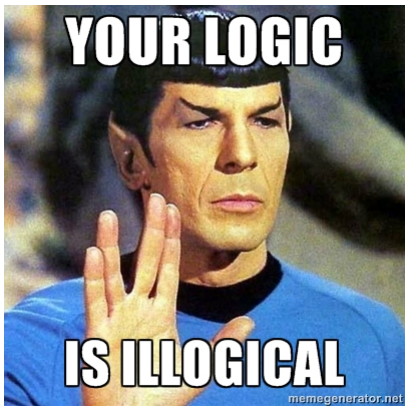
```
System.out.println("Enter 5 basketball scores:");
Scanner keyboard = new Scanner(System.in);
int score1 = keyboard.nextInt();
int score2 = keyboard.nextInt();
int score3 = keyboard.nextInt();
int score4 = keyboard.nextInt();
int score5 = keyboard.nextInt();
double average = (double) (score1 + score2 + score3 + score4 + score5) / 5.0;
System.out.println("Average score: " + average);

// repeat this for each of the 5 scores
if (score1 > average)
        System.out.println(score1 + ": above average");
else if (score1 < average)
        System.out.println(score1 + ": below average");
else
        System.out.println(score1 + ": equal to the average");

// if score2...score3...score4...
```

If we have more Scores…

- Think about 80 scores…
  - Declare 80 variables
  - Check them 80 times
- This is illogical!
- There must be an easier way!
  - What about things like: $Score_1$, $Score_2$, …, $Score_n$



Arrays

- `int[] scores = new int[5];`
- This is like declaring 5 strangely named variables of type int:
  - scores[0]
  - scores[1]
  - scores[2]
  - scores[3]
  - scores[4]
- Especially, you can use **score[i]** to locate a single one

Arrays

- An ***array*** is a collection of items of the same type
- Like a list of different variables, but with a nice, compact way to name them
- A special kind of object in Java
- **Loops repeat things temporally; arrays repeat things spatially**

Comparing Scores/Averages w/ Arrays

```
System.out.println("Enter 5 basketball scores:");
Scanner keyboard = new Scanner(System.in);
int[] scores = new int[5];
int scoreSum = 0;
for (int i = 0; i < 5; i++) {
        scores[i] = keyboard.nextInt();
        scoreSum += scores[i];
}
double average = (double) scoreSum / 5;
System.out.println("Average score: " + average);

for (int i = 0; i < 5; i++) {
        if (scores[i] > average)
                System.out.println(scores[i] + ": above average");
        else if (scores[i] < average)
                System.out.println(scores[i] + ": below average");
        else
                System.out.println(scores[i] + ": equal to the average");
}
```

Index

- Variables such as scores[0] and scores[1] that have an integer expression in square brackets are known as:
  - ***indexed variables***, ***subscripted variables***, ***array elements***, or simply ***elements***
- An ***index*** or ***subscript*** is an integer expression inside the square brackets that indicates an array element
  - ArrayName[index]

Index

- Where have we seen the word index before?
  - String's indexOf() method

H	o	w		a	r	e		y	o	u	?
0	1	2	3	4	5	6	7	8	9	10	11

  - str.indexOf('e') == 6;
  - str.charAt(6) == 'e';

Index

- Index numbers start with **0**. They do NOT start with 1 or any other number.
  - Not like counters in loops, you can't change the range of indices
- The reason is that the array name represents a memory address, and the i^{th} element can be accessed by the address plus i

Array and Index

var name	score[0]	score[1]	score[2]	score[3]	score[4]
data	62	51	88	70	74
m address	25131	25132	25133	25134	25135

- In history, computer scientists argued a lot on this
  - *"Should array indices start at 0 or 1? My compromise of 0.5 was rejected without, I thought, proper consideration."* – Stan Kelly-Bootle

Access Elements with Indices

- The number inside square brackets can be any integer expression
  - An integer: *scores[3]*
  - Variable of type int: *scores[index]*
  - Expression that evaluates to int: *scores[index*3]*
- Can use elements just like any other variables:
  - *scores[3] = 68;*
  - *scores[4] = scores[4] + 3;* // just made a 3-pointer!
  - *System.out.println(scores[1]);*

Indices and For-Loops

- In programming, a for-loop usually starts with counter i = 0. There is a reason

```
for (int i = 0; i < 5; i++) {
        scores[i] = keyboard.nextInt();
        scoreSum += scores[i];
}
```

Creating an Array

- Array is a special class and we create its objects
  - Syntax for creating an array:
    - **Base_Type[] Array_Name = new Base_Type[Length];**
  - Example:
    - **int[] pressure = new int[100];**
  - Alternatively:
    - **int[] pressure;**
    - **pressure = new int[100];**

Do not be OUT OF BOUNDS!

- Indices MUST be in bounds
  - double[] entries = new double[**5**]; // from [0] to [4]
  - entries[**5**] = 3.7; // ERROR! Index out of bounds
- Your code will compile if you are using an index that is out of bounds, but it will give you a run-time error!

Initializing Arrays

- You can initialize arrays when you declare them
  - int[] scores = { 68, 97, 102 };
- Equivalent to
  - int[] scores = new int[3];
  - scores[0] = 68;
  - scores[1] = 97;
  - scores[2] = 102;
- Or, you can use for-loop
  - When in doubt, for-loop!

Joke

- Q: Why did the programmer quit his job?
- A: Because he didn't get arrays.

  Hint: A raise ;-)

Next class

- More on arrays