COMP 110-003
Introduction to Programming
Arrays

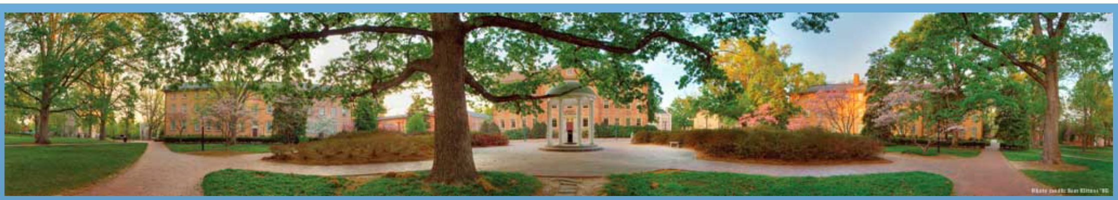
March 26, 2013

Haohan Li
TR 11:00 – 12:15, SN 011
Spring 2013

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

Announcement

- Program 4 is online!
- You will write a tic-tac-toe game



Requirements

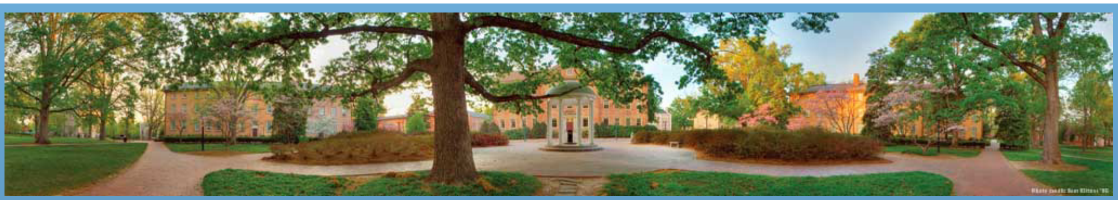
- The game can be displayed
  - When user clicks, a move will be made
- The game can be played
  - After the user moves, the program makes another move
- The game will end
  - The program judges if someone wins

Your Mission

- Write code to display things correctly
- Write code to judge if the game ends
- Write code to make automatic moves!
  - That’s what we call artificial intelligence
  - You can play against the AI I wrote and see if your program is smart enough

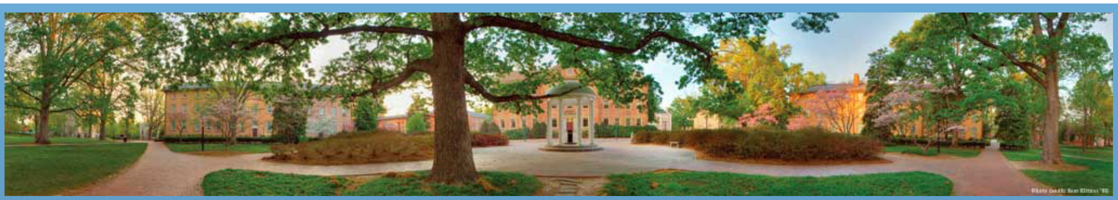
Milestones

- Submit something by April 11th
  - Do something before that time!
  - If you wait until the end of the semester, you are doomed
- Make it run and write a random “AI” by April 20th
  - You can submit things after April 20th and you lose points
- Write a smart AI and a report by April 30th
  - By that day, you must submit everything



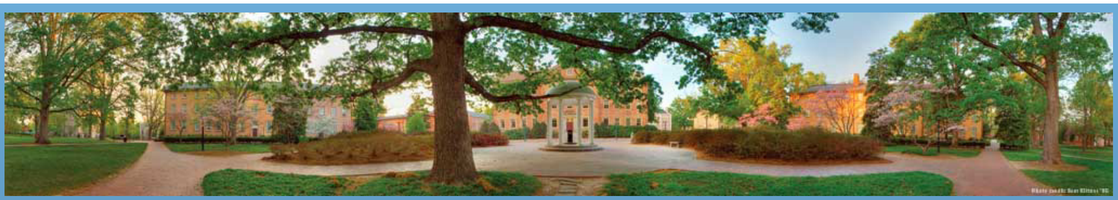
Start Immediately!!!

- Play the game online
  - Think about how to decide if a game wins/draws
  - Think about how to play smartly
  - Write down your algorithms
- You can start coding now, or wait until we learn more about arrays
  - Start coding no later than next week!
  - Code progressively; understand the structure first
- Ask questions if you have any – don't guess



Extra Points

- You must write your AI in a way that it doesn't rely on who moves first
  - Then your program will run if the computer moves first
- If you have this version, you can participate the tournament!
  - The AIs will fight each other
  - The winner will get extra points on the final grade



Daily Joke

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

Review

- Classes
- Objects
- Instance variables
- Methods
  - Return types
  - Parameters and arguments
- Information hiding and encapsulation
  - public/private
  - accessors/mutators

Class

```
public class Student {
     public String name;
     public int classYear;
     public double GPA;
     public String major;

// ...

     public String getMajor() {
          return major;
     }

     public void increaseYear() {
          classYear++;
     }
}
```

Class name

Data
(or attributes, or
instance variables)

Methods

Using a Class

```
public class Student {
     public String name;
     public int classYear;
     public double GPA;
     public String major;

// ...

     public String getMajor() {
          return major;
     }

     public void increaseYear() {
          classYear++;
     }
}
```

```
public class StudentTest {
  public static void main(String[] args) {
     Student jack = new Student();
     jack.name = "Jack Smith";
     jack.major = "Computer Science";
     jack.classYear = 1;
     jack.GPA = 3.5;

     String m = jack.getMajor(); //
     System.out.println("Jack's major is " + m);

     jack.increaseYear();

     System.out.println("Jack's class year is now
" + jack.classYear);

  }
}
```

Methods

```
public class Student
{
    private String name;
    private int age;

    public void setName(String studentName) {
        name = studentName;
    }
    public void setAge(int studentAge) {
        age = studentAge;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
}
```

Mutators

Accessors

Methods with Parameters

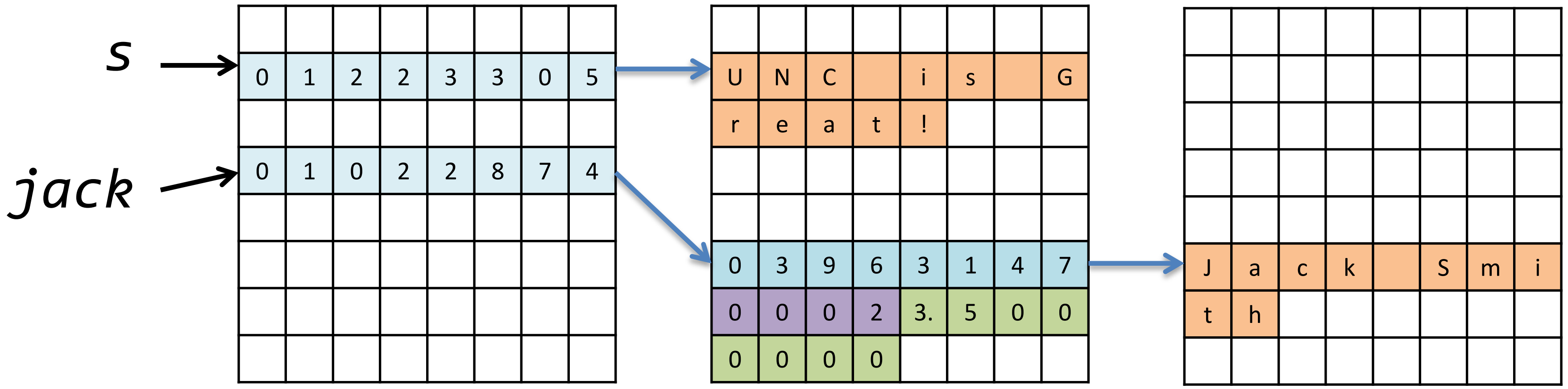
- Parameters are used to hold the value that you pass to the method
- Parameters can be used as (local) variables inside the method

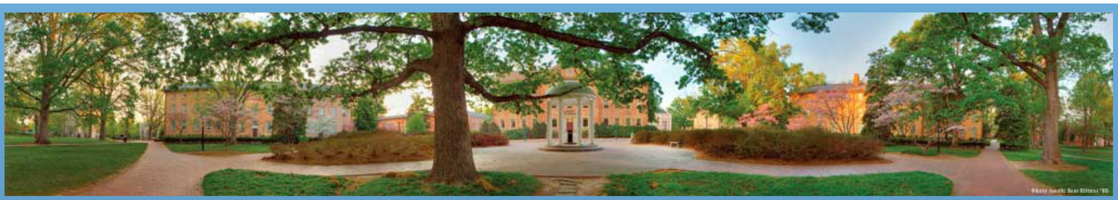
```
public int square(int number)
{
    return number * number;
}
```

Parameters go inside the parentheses of method header

Variables of a Class Type

- What goes in these variables?
  - In a class type variable, the address pointing to the actual object is saved (not the object itself)





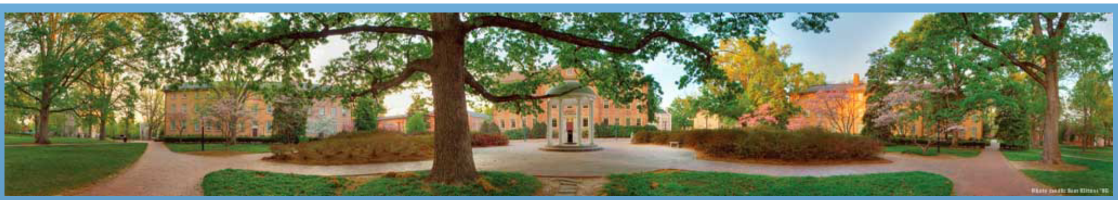
Arrays

- To think about arrays, let’s think about loops first
- Why do we need loops?
  - Because we want to repeat things without write them again and again
  - Think about the average score problem

Average Score without Loops

- Assuming that we only need 5 scores

```
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

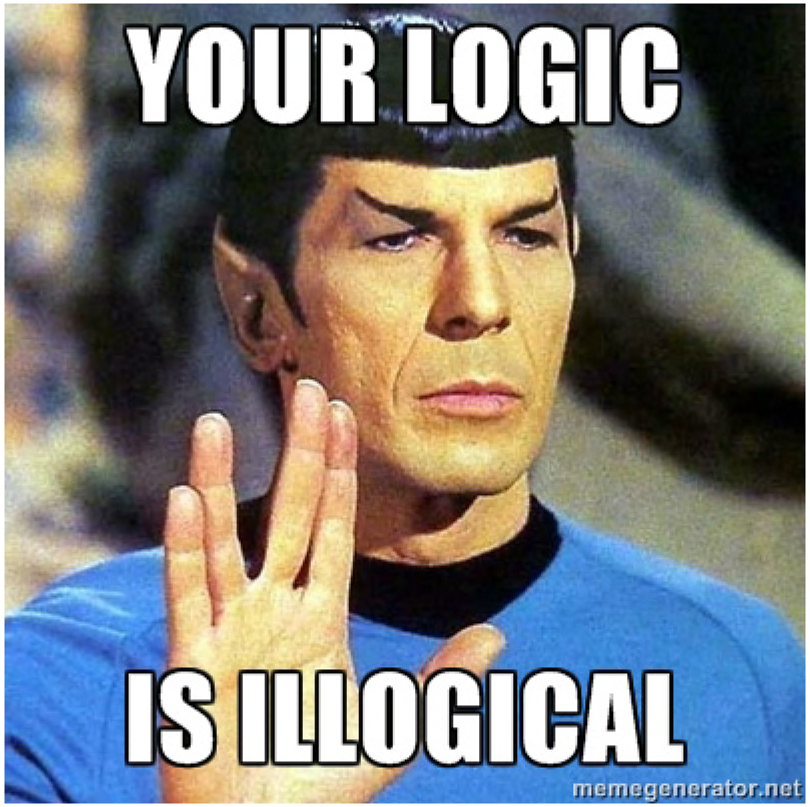
- If we really need to save these scores, loop won’t help you
- Think about the requirement
  - Print out if a score is above/below average
  - We have to calculate average first, then decide if a score is above/below average
  - Therefore we must save all these scores, and compare them to the average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

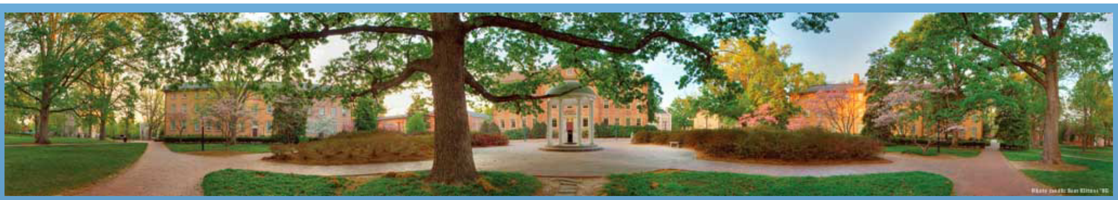
Comparing All Scores and the Average

```java
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/Average 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

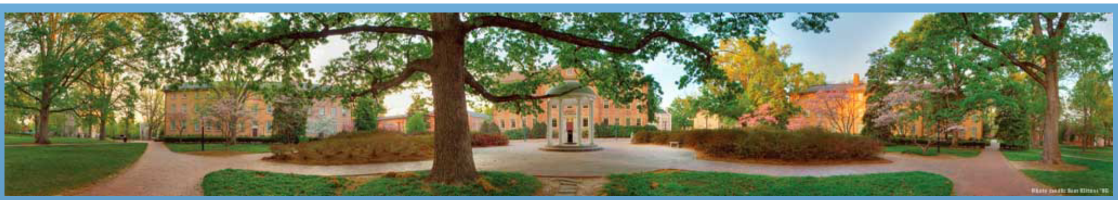
- Variables such as scores[0] and scores[1] that have an integer expression in square brackets are known as:
  - ***indexed variables***, ***subscripted variables***, ***array elements***, or simply ***elements***
- An ***index*** or ***subscript*** is an integer expression inside the square brackets 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’;
  - char[] ca = str.toCharArray();
  - char[6] == ‘e’;
- In C, there is only char arrays instead of Strings(FYI)

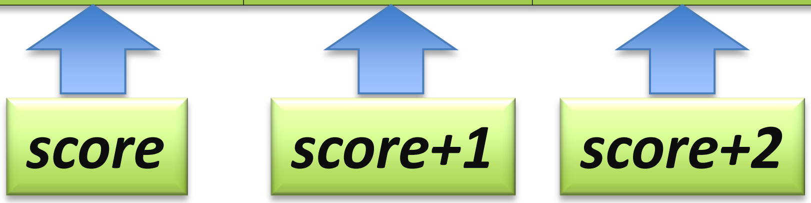


Index

- 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];
}
```

The Example Again

```
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");
}
```

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];**

Creating an Array

- The base type can be any type
  - **double**[] temperature = new **double**[7];
  - **Student**[] students = new **Student**[35];
- The number of elements in an array is called its **length** or **size**
  - temperature has 7 elements, temperature[0] through temperature[6]
  - students has 35 elements, students[0] through students[34]

Creating an Array

- Create an array with given length saved in constants
  - **public static final int NUMBER_OF_READINGS** = 100;
  - **int**[] pressure = **new int**[**NUMBER_OF_READINGS**];
- Create an array with user input length
  - System.out.println("How many scores?");
  - **int numScores** = keyboard.nextInt();
  - **int**[] scores = **new int**[**numScores**];

Finding Length of An Existing Array

- An array is a special kind of object
  - It has one public instance variable: *length*
  - *length* is equal to the length of the array

    ```
    Pet[] pets = new Pet[20];
    pets.length has the value 20
    ```

  - You cannot change the value of *length* because it is **final**

The Example Again (and again…)

```
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 < scores.length; i++) {
     scores[i] = keyboard.nextInt();
     scoreSum += scores[i];
}
double average = (double) scoreSum / 5;
System.out.println("Average score: " + average);

for (int i = 0; i < scores.length; 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");
}
```

Don’t 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!