CSCI 136
Data Structures & Advanced Programming

Spring 2020

Bill Jannen & Dan Barowy

Administrative Details

- Class roster: Who's here?
  - And who's trying to get in (there's room!)?
- Handouts: syllabus, honor code, textbook
- Lecture location: Schow 030B
- Lab: Wed 12-2pm or 2-4pm
- Lab location: TCL 217a & 216
- Lab entry code: 0-1-2-4-8-16 (memorize now!)
- Course Webpage (updated regularly…):

  http://cs.williams.edu/~cs136

Announcements

- Colloquium Today: Department Research
  - Professors will be presenting their research topics
  - Good chance to see what we do, and…
  - Helps you decide if you want to do summer research!
- Cognitive Science candidate talk on Monday, Schow 030A @4pm
  - Joanna Morris: "How we Read Complex Words"

Today's Outline

- Course Preview
- Course Bureaucracy
- Java (re)fresher–Hello World(s)

Why Take CS136?

- To learn about:
  - Data Structures
    - Effective ways to store and manipulate data
  - Advanced Programming
    - Use structures and techniques to write programs that solve interesting and important problems
  - Basics of Algorithm Analysis
    - Measuring algorithm complexity
    - Determining algorithm correctness

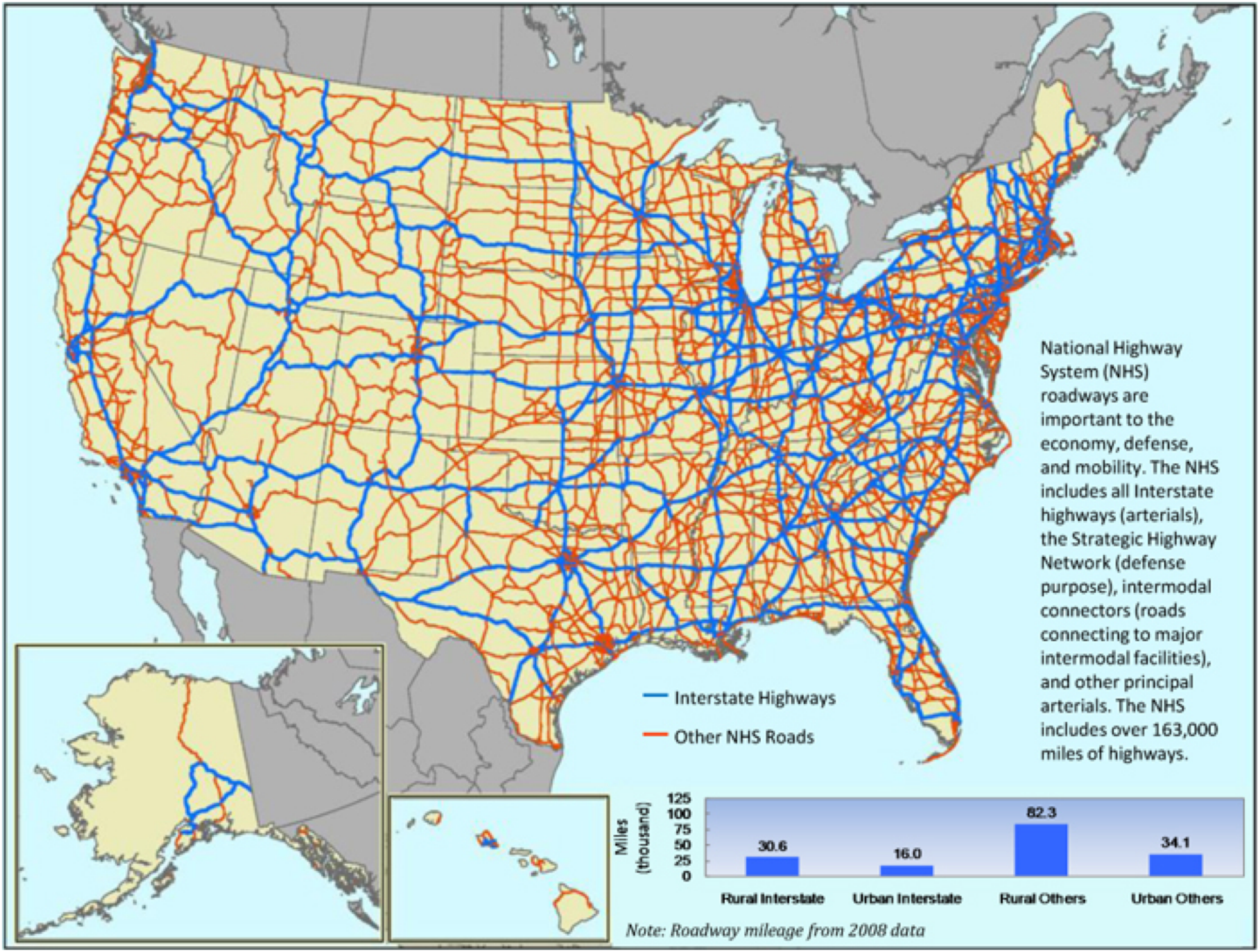
Course Goals

- Identify basic data structures
  - list, stack, array, tree, graph, hash table, and more
- Implement these structures in Java
- Learn how to evaluate and visualize data structures
  - Different representations of the data
  - Different algorithms for manipulating/accessing/storing data
  - E.g., linked lists and arrays both represent lists of items
- Learn how to design larger programs that are easier to modify, extend, and debug
- **Have fun!**

Common Themes

1. Identify data for a problem
2. Identify questions to answer about data
3. Design data structures and algorithms to answer questions *correctly* and *efficiently* (Note: not all correct solutions are efficient, and vice versa!)
4. Implement solutions that are robust, adaptable, and reusable

Example: Shortest Paths in Networks


Interstate Highways
Other NHS Roads
National Highway System (NHS) roadways are important to the economy, defense, and mobility. The NHS includes all Interstate highways (arterials), the Strategic Highway Network (defense purpose), intermodal connectors (roads connecting to major intermodal facilities), and other principal arterials. The NHS includes over 163,000 miles of highways.
Miles (thousand)
125
100
75
50
25
0
30.6
16.0
82.3
34.1
Rural Interstate
Urban Interstate
Rural Others
Urban Others
Note: Roadway mileage from 2008 data

Finding Shortest Paths

- What is "the data"?
  - Road segments: Source, destination, length (weight)
- What is the "question"?
  - Given source and destination, compute the shortest path from source
- What is the algorithm? Dijkstra's Algorithm
- What are the data structures? (spoiler alert!)
  - Graph: holds the road network in some useful form
  - Priority Queue: holds not-yet-inspected edges
  - Also uses: Lists, arrays, stacks, ...
- A quick demo….

Course Outline

- Java overview
- Core data structures
  - Vectors (extensible arrays), lists, queues, stacks
- Advanced data structures
  - Trees, heaps, graphs, hashtables
- Foundations (throughout semester)
  - Vocabulary
  - Analysis tools
  - Recursion & Induction
  - Methodology

Syllabus Highlights

- How to contact us
  - Bill Jannen (TCL 306)
    - Office hours: M: 1-2pm, F: 4-5pm, and by appointment
    - mailto:jannen@cs.williams.edu
  - Dan Barowy (TCL 307)
    - Office hours: M: 4-5pm, F: 4-5pm, and by appointment
    - mailto:dbarowy@cs.williams.edu
  - Piazza – Please, please, PLEASE post your questions
- Textbook
  - Java Structures: Data Structures in Java for the Principled Programmer, √7 Edition (by Duane Bailey)
  - Take one: You're already paying for it!
- Weekly labs and quizzes, mid-term & final exam....

Syllabus Highlights

- Quizzes
  - Monday: ungraded quiz on textbook material
  - Friday: graded quiz (twist on Monday's material)
- Labs
  - Every Wednesday
  - Due Mondays at 8pm
  - Deadline is firm, but…
- Resubmissions
  - 2 per semester
  - Can earn back up to 50% of missed points
  - See syllabus for format & restrictions

Syllabus Highlights

- Code review
  - Labs graded on correctness, design, and style
  - But it is sometimes hard to intuit good design and style
- Lida has dedicated slots for code review
  - All you must do is attend and discuss: not graded twice for your code
  - Must sign up for one slot during the semester
  - Earlier is probably more helpful, later probably more substantive discussion…
- If slots don't fill in a given week, we may reach out to you: don't read into it!
  - 60 students means we need to fill the slots if we want to get to everyone…

Honor Code and Ethics

- College Honor Code and Computer Ethics guidelines can be found here:
  - https://sites.williams.edu/honor-system/
  - https://oit.williams.edu/policies/ethics/
- You should also know the CS Department computer usage policy.
  - https://csci.williams.edu/the-cs-honor-code-and-computer-usage-policy/
  - If you are not familiar with these items, please review them.
- Review the handout and individual lab details
- We take these things very seriously…

Your Responsibilities

- Come to lab and lecture on time
- Read assigned material before class and lab
  - Bring textbook to lab (or be prepared to use PDF)
  - Bring paper/pen(cil) to lab for brain-storming, … PPP
- **Come to lab prepared**
  - Bring design docs for program
  - 1 Prof + 1TA == help for you: take advantage of this
- Do NOT accept prolonged confusion! Ask questions
- Your work should be your own. Unsure? Ask!
- Participate: discussion, Piazza, office hours, etc.

Accounts and Passwords

- Before the first lab
  - Login to your CS Mac Lab account (different than OIT!!!)
  - If you don't have an account, see Mary or Lida
  - If you forgot your password, see Mary or Lida
- Mary and Lida manage our systems.
  - Mary's office is in the 3rd floor CS lab (TCL 312)
  - Lida's office is TCL 205
- Get this sorted out before lab on Wednesday!
  - "Office hours" Lida: 2/7 3:30-4:15pm, Mary: 2/10 2-4pm
- Complete Pre-lab: Step 0 by Monday 4pm
- Complete "Getting to know each other" survey

Why Java?

- There are lots of programming languages…
  - C, LISP, C++, Java, C#, Python
- Java was designed in 1990s to support Internet programming
- Why Java?
  - It's easier (than predecessors like C++) to write correct programs
  - Object-oriented – good for large systems
  - Good support for abstraction, extension, modularization
  - Automatically handles low-level memory management
  - Very portable

Why Not BlueJ?

- Learn to use Unix
  - Command-line tools
  - Emacs: a standard Unix-based editor
  - Atom: a customizable featureful text editor
- Emphasis will move from user interface programming to data structures and efficient algorithm design
- Take advantage of opportunity to become Unix-savvy!

Java Crash Course

Simple Sample Programs

- Hello.java
  - Write a program that prints “Hello” to the terminal.
  - Now let’s run it.
- Of Note:
  - `public static void main(String[] args){...}`
  - System.out is of type PrintStream
  - javac and java commands
  - Terminal.app

Sample Programs

- Sum0-5.java
  - Programs that adds two integers
- Of Note:
  - System.in is of type ReadStream
  - Scanner class provides parsing of text streams (terminal input, files, Strings, etc)
  - args[] is passed to `main` from the OS environment
    - args[] contains command-line arguments held as Strings
  - Integer.valueOf(...) converts String to int
  - Static values/methods: in, out, valueOf, main

Java Reference Materials!

Please see *Bailey Appendix B* for a good Java Reference. The following slides show some examples as a "refresher" but are not intended to be exhaustive…

Java Review (See Appendix B)

- Variable types
  - Primitive: int, double, boolean, ...
  - Object (class-based): String (special), Point, Jbutton, ...
  - Arrays

Java Review (See Appendix B)

- Statements
  - `int x;  // declare variable x`
  - `int x = 3; // declare & initialize x`
  - `x = x + 1;`
  - `x++;`
  - `if (x > 3) { … } else { … }`
  - `while (x < 2) { … }`
  - `for (int i = 0; i < x; i++) { … }`

Java Review (See Appendix B)

- Comments
  - `// this is a single-line comment`
  - `/* this can span multiple lines */`

- Aside: *good* comments make code readable
  - Explain the "why" not the "what"
  - State assumptions or non-obvious logic

```
return  x+1; // returns sum of x+1
while (y < 2) /* continue as long
                 * as y is < 2
                 */
```

Primitive Types

- Provide numeric, character, and logical values
  - 11, -23, 4.21, 'c', false
- Can be associated with a name (*variable*)
- Variables *must* be declared before use

```
int age;      // A simple integer value
float speed; // A number with a 'decimal' part
char grade;  // A single character
bool loggedIn; // Either true or false
```

- Variables *can* be initialized when declared

```
int age = 21;
float speed = 47.25;
char grade = 'A';
bool loggedIn = true;
```

Array Types

- Holds a collection of values of some type
- Can be of any type

```
int[] ages;         // An array of integeras
float[] speeds;     // An array of floats
char[] grades;      // An array of characters
bool[] loggedIn;    // Either true or false
```

- Arrays can be initialized when declared

```
int[] ages = { 21, 20, 19, 19, 20 };
float[] speeds = { 47.25, 3.4, -2.13, 0.0 };
char[] grades = { 'A', 'B', 'c', 'C' };
bool[] loggedIn = { true, true, false, true };
```

- Or just created with a standard default value

```
int[] ages = new int[15]; // array of 15 0s
```

"Everything is a class"

- Typically put the code for each class in a file with the same name as the class
  - The `Person` class' code would be in `Person.java`
- The method `'main'` is the entry point to a Java program
  - `main` has a specific method signature:

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

- In grand CS tradition, we will write and run `Hello.java`

Operators

Java provides a number of built-in *operators* including

- Arithmetic operators: +, -, *, /, %
- Relational operators: ==, !=, <, **≤, >, ≥**
- Logical operators &&, || (don't use &, |)
- Assignment operators =, +=, -=, *=, /=, ...

Common unary operators include

- Arithmetic: -, ++, -- (prefix and postfix)
- Logical: ! (not)

Operator Precedence in Java

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Operator Gotchas!

- There is no exponentiation operator in Java.
  - The symbol ^ is the *bitwise or* operator in Java.
- The *remainder* operator % is the same as the mathematical 'mod' function for *positive* arguments,
  - For negative arguments it is not: -8 % 3 = -2
- The logical operators && and || use *short-circuit evaluation*:
  - Once the value of the logical expression can be determined, no further evaluation takes place.
  - E.g.: If n = 0, then (n != 0 && (k/n > 3), will yield false without evaluating k/n. Very useful!

Expressions

Computations described by applying operators to other values (variables, literals, values returned from method calls)

- An expression returns a value
  - `3+2*5 - 7/4  // returns 12`
  - `x + y*z – q/w`
  - `(- b + Math.sqrt(b*b – 4 * a * c) )/( 2* a)`
  - `( n > 0) && (k / n > 2) // computes a` *`boolean`*
- Assignment expression: x = 3; // returns 3
  - So y = 4 * (x = 3) sets x = 3 and y = 12 (and returns 12)

Boolean expressions let us control program *flow of execution* when combined with *control structures*

Control Structures

Select next statement to execute based on value of a boolean expression. Two flavors

- Looping structures: while, do/while, for
  - Repeatedly execute same statement (block)
- Branching structures: if, if/else, switch
  - Select one of several possible statements (blocks)
  - Special: break/continue: exit a looping structure
    - break: exits loop completely
    - continue: proceeds to next iteration of loop

while & do-while

Compare this...

```
Random rng = new Random();
int flip = rng.nextInt(2), count = 0;
while (flip == 0) {     // count flips until “heads”
      count++;
      flip = rng.nextInt(2);
}
```

...to this

```
int flip, count = 0;
do {                    // count flips until “heads”
      count++;
      flip = rng.nextInt(2);
} while (flip == 0) ;
```

For & for-each

Here's a typical **for** loop example

```
int[] grades = { 100, 78, 92, 87, 89, 90 };
int sum = 0;
for( int i = 0; i < grades.length; i++ ) sum +=
grades[i];
```

This **for** construct is equivalent to

```
int i = 0;
while ( i < grades.length ) {
    sum += grades[i];
    i++;
}
```

Can also write

```
for (int g : grades ) sum += g;
// called for-each construct
```

Loop Construct Notes

- The body of a **while** loop may not ever be executed
- The body of a **do – while** loop always executes at least once
- **For** loops are typically used when number of iterations desired is known in advance. E.g.
  - Execute loop exactly 100 times
  - Execute loop for each element of an array
- The **for-each** construct is often used to access array (and other collection type) values when *no updating* of the array is required
  - We'll explore this construct more later in the course

If/else

```
if (x > 0)          // There is exactly 1 "if" clause
      y = 1 / x;
else if (x<0) {     // 0 or more "else if" clauses
      x = - x;
      y = 1 / x;
}
else                // at most 1 "else" clause
      System.out.println("Can't divide by 0!");
```

The single statement can be replaced by a *block*: any sequence of statements enclosed in {}

switch

Example: Encode clubs, diamonds, hearts, spades as 0, 1, 2, 3

```
switch (x) {
      case 0: case 2:
            System.out.println("Your card is red");
            break;
      case 1: case 3:
            System.out.println("Your card is
black");
            break;
      default:
            System.out.println("Illegal suit
code!");
            break;
}
```

Break & Continue

Find first prime > 100

```
for( int i = 101; ; i++ )
    if ( isPrime(i) ) {
          System.out.println( i );
          break;
    }
```

Print primes < 100

```
for( int i = 1; i < 100 ; i++ ) {
    if ( !isPrime(i) )
          continue;
    System.out.println( i );
}
```

Summary

Basic Java elements so far

- Primitive and array types
- Variable declaration and assignment
- Operators & operator precedence
- Expressions
- Control structures
  - Branching: if – else, switch, break, continue
  - Looping: while, do – while, for, for – each
- Edit (emacs), compile (javac), run (java) cycle