Course Staff

- **Instructors:**
  - Professor Andrew Myers
  - Professor David Schwartz

- **Administrative Assistant:**
  - Kelly Patwell

- **Locations, office hours, contact info?**
  - See [Staff](http://www.cs.cornell.edu/courses/cs211) on website
Student Course Staff

• **Teaching Assistants:**
  – TAs lead recitation sections (starting today)
  – TAs are your main contact point

• **Consultants:**
  – In Upson 360, hours TBA online
  – “Front line” for answering questions

• **More info?**
  – See [Staff](#) on website

---

Lectures

• TR 10:10-11am, Olin 155
• Attendance is mandatory
• Lecture notes will be online—print them before class and bring them to class
• We will occasionally make small last minute changes to the notes, so don’t print them too far in advance
• Readings will be posted online together with lecture notes
CS212

- **CS 212: Java Practicum**
- 1 credit project course
- Substantial project
- 1 lecture per week
- Required for CS majors; recommended for others
- Take 211 and 212 in same semester?
Online resources

• Course web site
  http://www.cs.cornell.edu/Courses/cs211
  – Watch for announcements
• Course newsgroup
  cornell.class.cs211
  – Good place to ask questions (carefully)
• Textbook: Weiss, “Data Structures and Problem Solving Using Java”

Obtaining Java

• We do not require an IDE
  – But we generally use Eclipse
• See Help & Software under Java Resources on website
Java Help

• CS 211 assumes basic Java knowledge:
  – control structures
  – arrays, strings
  – classes (fields, methods, constructors)

• Need review?
  – Tutorials, links on website (Help & Software)
  – Java Refresher/Bootcamp:
    • self-guided tutorial—material on website
    • You can also work with staff on it: 7:30-10:30pm on both Thu 1/26 and Tue 1/31 in Upson B7
    • Same material on both days

Academic Excellence Workshops

• Two-hour labs in which students work together in cooperative setting
• One credit S/U course based on attendance
• ENGRG 210, 745-791, Fridays, 2:30-4:25, ACCEL
• See CS211 web site for more info
Course Work

• 6 assignments involving both programming and written answers
  – We A.I. check each homework assignment
• Two prelims and final exam
• Course evaluation

<table>
<thead>
<tr>
<th>Assignments (44%)</th>
<th>Exams (55%)</th>
<th>Eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>A2</td>
<td>A3</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

Assignments

• Assignments may be done by teams of two students (except A1)
• You can do them by yourself if you like
• Finding a partner: choose your own or contact your TA. Newsgroup may be helpful.
• Monogamy is strongly encouraged, polygamy is strongly discouraged, and divorces are permitted in case of irreconcilable differences
• See website course info and Code of Academic Integrity
CS211 Objectives

An introduction to computer science and software engineering

• Concepts in modern programming languages:
  – recursive algorithms and data structures
  – data abstraction, subtyping, generic programming
  – frameworks and event-driven programming

• Analyzing, designing for efficiency
  – asymptotic complexity, induction

• Data structures and algorithms: arrays, lists, stacks, queues, trees, hash tables, graphs

• Organizing large programs

Using Java, but not a course on Java!

Lecture Sequence

• Introduction and Review
• Recursion and induction
• Object-oriented concepts: data abstraction, subtyping
• Data structures: Lists and trees
• Grammars and parsing
• Inheritance and frameworks
• Algorithm analysis
  – Asymptotic Complexity
• Searching and Sorting
More Lecture Topics

- Generic Programming
- Abstract Data Types
  - Sequence Structures: stacks, queues, heaps, priority queues
  - Search Structures: binary search trees, hashing
  - Graphs and graph algorithms
- Graphical user interface frameworks
  - Event-driven programming
  - Concurrency and simple synchronization

Sam Loyd’s 8 Puzzle

Goal: Given an initial configuration of tiles, find a sequence of moves that will lead to the sorted configuration.

A particular configuration is called a state of the puzzle.
State Transition Diagram of 8-Puzzle

State Transition Diagram: picture of adjacent states. A state Y is adjacent to state X if Y can be reached from X in one move.

State Transition Diagram for a 2x2 Puzzle

Sorted State

Solutions for this state:
- SWN
- WSENWSENW
- SWEWN

---

9
Graphs

• State Transition Diagram in previous slide is an example of a graph: a mathematical abstraction
  – vertices (or nodes): (e.g., the puzzle states)
  – edges (or arcs): connections between pairs of vertices
  – vertices and edges may be labeled with some information (name, direction, weight, cost, …)

• Other examples of graphs: airline routes, roadmaps, . . .
  – A common vocabulary for problems

Path Problems in Graphs

• Is there a path from node A to node B?
  – Solve the 8-puzzle

• What is the shortest path from A to B?
  – 8-puzzle (efficiently), Mapquest, …

• Traveling salesman problem
• Hamiltonian cycles
• . . . will see later
Simulating 8-puzzle

• What operations should puzzle objects support?
• How do we represent states?
• How do we specify an initial state?
• What algorithm do we use to solve a given initial configuration?
• What kind of GUI should we design?
• How to structure the program so it can be maintained, fixed, upgraded?

SaM

• *SaM* is a simple StAck Machine:
  – Similar to the Java Virtual Machine (JVM)
  – and to the machine code understood by processor hardware
  – Use it to understand how compilers work
• Download it from course homepage
• Used extensively in CS212
SaM’s Stack

Stack: an array of integers
- Stack grows when integer is "pushed" on top.
- Stack shrinks when integer is "popped" from top.
- Stack starts at address 0 and grows to larger addresses.

Stack pointer (SP):
- first "free" address in stack
- stores integer address
- initialized to 0

Note: For now, assume only integers can be pushed on stack. SaM actually allows floats, characters, etc. to be pushed, and it tracks type of data. GUI displays type (I:integer, F:float, . . . ), but ignore this for now.

Some SaM Commands

- All arithmetic/logical operations pop values from stack, perform operation, push result, and move SP to first free address
- Some commands:
  ```
PUSHIMM int
  // push integer int onto top of stack
ADD
  // pops two values from top of stack
  // adds them and pushes result
SUB
  // pops two values (say top and below)
  // and pushes result of doing (below - top)
TIMES
  // works like ADD
GREATER
  // Boolean values are simulated using 0/1 (false/true)
AND
  // logical AND
STOP
  // terminate execution of program
```
Demonstrate SaM Commands

Booleans and SaM

Booleans are simulated in SaM with integers:
• False → 0
• True → any int except 0 (usually 1)

GREATER:
- Pop two values ($V_{\text{top}}$ and $V_{\text{below}}$) from stack (V).
- So, $V_{\text{top}} = 7$ and $V_{\text{below}} = -2$.
- If $V_{\text{below}} > V_{\text{top}}$, push 1; else push 0.
- In example, we would push 0.
SaM Programs

• Example 1:
  \begin{verbatim}
PUSHIMM 5
PUSHIMM 4
PUSHIMM 3
PUSHIMM 2
TIMES
TIMES
TIMES
STOP // should leave 120 on top of stack
\end{verbatim}

• Example 2:
  \begin{verbatim}
PUSHIMM 5
PUSHIMM 4
GREATER
STOP // should leave 1 on top of stack
\end{verbatim}

SaM Simulator

• What operations must SaM objects support?
• How do we represent the internal state of SaM?
• How do we load programs from a file?
• How do we write code to interpret each of the opcodes?
• How do we turn a high-level language like Java into SaM code?
• See “Chapter 1” in CS212 lecture notes
Why you need CS 211

You will be able to design and write moderately large, well-structured programs to simulate such systems. *Useful because:*

1. **Computer systems are complex. Need CS to make them work; can’t just hack it**
   - Selected software disasters:
     - CTAS air traffic control system 1991-present
     - Ariane 5 ex-rocket
     - Denver airport automated baggage handling

Why you need CS211, cont’d

2. **Fun and intellectually interesting: cool math ideas meet engineering and make a difference.**
   - Recursion, induction, logic, discrete structures, …

3. **Crucial to any engineering or science career**
   - Good programmers are >10x more productive
   - Leverages knowledge in other fields, makes new possibilities
   - Where will you be in 10 years?
Moore’s Law

Grandmother’s Law

- Brain takes about 0.1 second to recognize your grandmother
  - About 1 second to add two integers (e.g. 3+4=7)
  - About 10 seconds to think/write statement of code
- Your brain is not getting any faster!
Motivation

• **Computers double in speed every 18 months**
  – Software doubles in size every M Years
  – Data doubles in size every N Years
  – Your brain never doubles in speed
  – But we do get smarter, and can work in teams
• **Computer science is increasingly important**
  – Better algorithms
  – Better data structures
  – Better programming languages
  – Better understanding of what is (and is not) possible