# Lecture 1

## Overview of CS/ENGR 211

http://www.cs.cornell.edu/Courses/cs211/2003fa



## Course Staff

Instructors:

- Professor Keshav Pingali
- Professor David Schwartz

TA's:

- Each TA will lead one or two recitation sections.
- Your TA is your main point of contact for the course: get to know him/her well.

Consultants: in Upson 304

Office hours: TBA online

Course Administrator: Helene Croft in Upson 5146



## Lectures

- TR 10:10-11:00 AM, Olin 155
- Attendance is mandatory
- Lecture notes will be online. Print them before class and bring them to class.
- Readings will be posted online together with lecture notes.



## Sections

- 10 sections
- Each section will be led by someone on staff
- Sections may cover material not covered in class: you must show up
- Pick one section and attend it
- No permission needed to switch sections

## CS 212

- 1 credit project course
- substantial project
- 1 lecture per week
- required for CS majors
- strongly advised to take 211 and 212 in same semester



## Java Bootcamp

- CS 211 assumes basic Java knowledge: classes, objects, methods, instance variables
- Students with little or no Java knowledge: attend Java bootcamp
- Bootcamp will be taught by Professor Schwartz
- Time and place: Upson B17, September 2/4 (TR), 7:30PM-10:30PM



## Academic Excellence Workshops

- Two-hour labs in which students work together in co-operative setting.
- One credit S/U course based on attendance.
- See course homepage for details.



## Coursework

- 6 assignments involving both programming and written answers: 42% of grade
- Exercises: 3%
- Two prelims: 15% of grade each
- Final exam: 25% of grade
- These weights may change.

## Assignments

- Assignments may be done by teams of two students.
- You can do them by yourself if you like.
- Finding a partner: post to newsgroup or email dis@cs.cornell.edu
- Monogamy is strongly encouraged, polygamy/polyandry is strongly discouraged, and divorces are permitted in case of irreconcilable differences.
- See syllabus and code of academic integrity online.



## Objectives of CS 211

Learn the following.

- Concepts in modern programming languages:
  1. recursion, induction
  2. classes, objects
  3. inheritance, interfaces
- Efficiency of programs
- Data structures: arrays, lists, stacks, queues, trees, hash-tables, graphs.
- Software engineering: How to organize large programs

This is not a course on Java programming.



## Lecture Sequence

- Ur-Java: simple non-OO language
- Induction and recursion
- Overview of OO programming
- Interfaces
- Lists and Trees
- Inheritance
- Searching and sorting
- Asymptotic complexity
- Inner classes



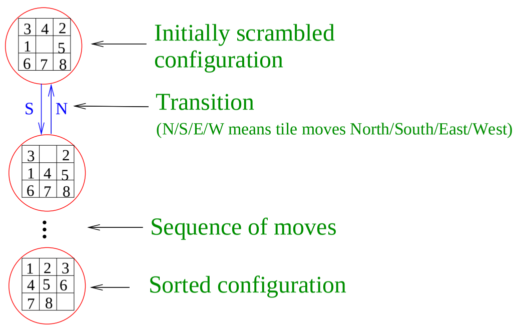
- Sequence structures
  - Stacks
  - Queues
  - Priority queues
- Search structures
  - Hash tables
  - Binary search trees
- Graphs and graph algorithms

Course is organized around concrete examples.

- Game of 8-puzzle
- Virtual machine: SaM
- others...



## Sam Loyd's 8-puzzle

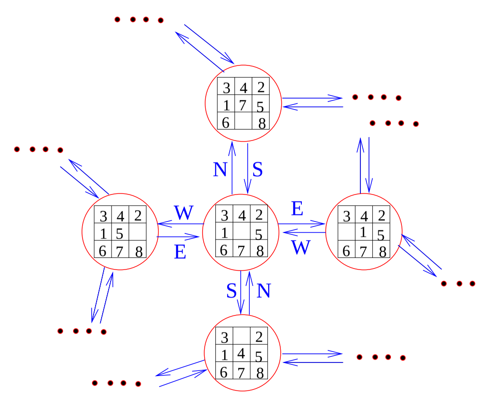



Goal: given an initial configuration of tiles, find a sequence of moves that will lead to the sorted configuration

A particular configuration is called a STATE of the puzzle.



## State Transition Diagram of 8-puzzle

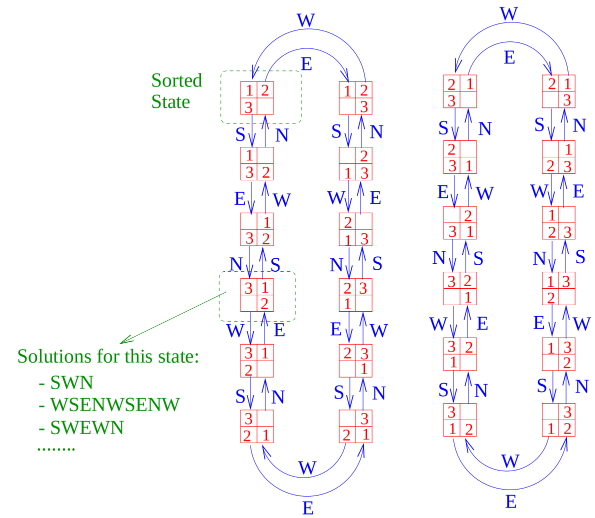


A state Y is ADJACENT to state X if Y can be reached from X in one move

State Transition Diagram: picture of adjacent states



## State Transition Diagram for a 2x2 Puzzle



## Graph

- State Transition Diagram in previous slide is an example of a GRAPH.
- Graph has
  - NODES: in our example, these are the puzzle states
  - EDGES: connections between pairs of nodes.
  - nodes and edges may be annotated with some information.
- Other examples of graphs: airline routes, roadmaps, ...
- Path problems in graphs:
  - Is there a path from node A to node B?
  - What is the shortest path from A to B?
  - Traveling Salesman's problem
  - Hamiltonian cycles
  - ....see later in semester



## Writing code for simulating 8-puzzle

1. What operations should puzzle objects support?
2. How do we represent configurations?
3. How do we specify an initial configuration?
4. What algorithm do we use to solve a given initial configuration?
5. What kind of GUI makes sense for puzzles?



## What is SaM?

SaM is a simple StAck Machine:

(i) Modeled roughly after the Java Virtual Machine (JVM).

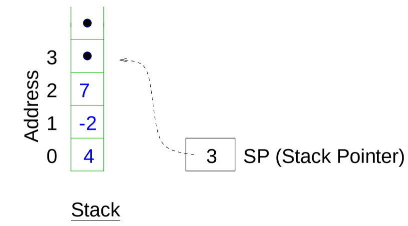
(ii) Use it to understand how computers work at the assembly language level (.class file level)

(iii) Use it to understand how compilers work

(iv) You can download it from course homepage



## Heart of SaM: a Stack and Stack Pointer (SP)

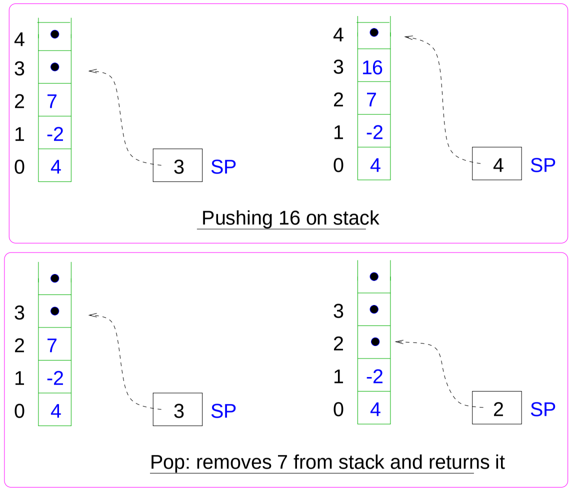



Stack: an array of integers
Stack grows when integer is "pushed" on top.
Stack shrinks when integer is "popped" from top.
Stack starts at address 0 and grows to larger addresses.

Stack pointer: first "free" address in stack
(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 will display type (I:integer,F:float,...), but ignore this for now.

## Operations on Stack




Pushing an Integer

Popping an Integer

Stack operations are used to implement SaM commands.
They are NOT SaM commands themselves.



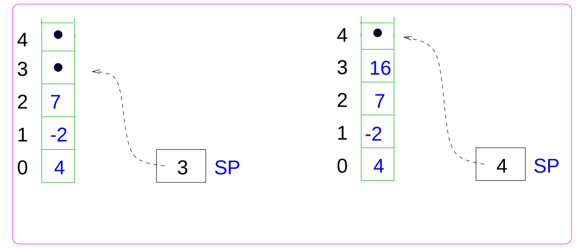
## SaM commands

ALL arithmetic/logical operations pop values from stack perform operation and push result.

```
PUSHIMM     *some integer*
        //pushes that integer on 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
GREATER
       // boolean values are simulated using 0/1 (false/true)
AND
       //logical AND
..........
STOP  //terminate execution of program
```



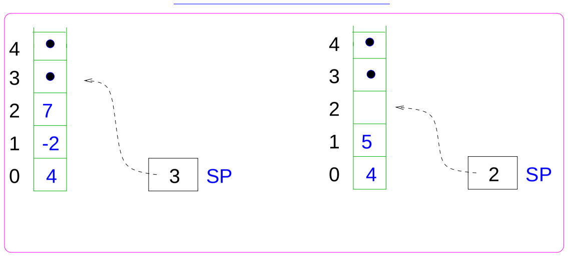
## SaM Commands




PUSHIMM 16

- push the integer 16 on stack

## SaM Commands



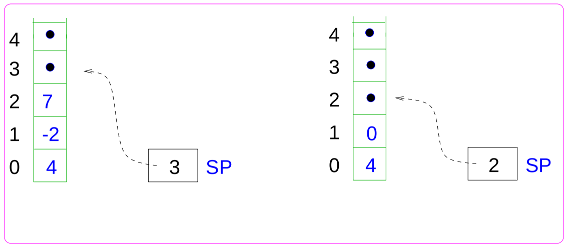

ADD

- pop two values from stack (7 and -2)
- add them (5)
- push result

SUB: similar; result would be (-2) - (7) = -9



## SaM Commands




Booleans are simulated in SaM with integers
True -> 1, false -> 0

GREATER

- pop two values (Vtop and Vbelow) from stack
- in example, Vtop = 7 and Vbelow = -2
- if Vbelow > Vtop push 1 else push 0
- in example, we would push 0.



Here are two simple SaM programs:

```
PUSHIMM 5
PUSHIMM 4
PUSHIMM 3
PUSHIMM 2
TIMES
TIMES
TIMES
STOP //should leave 120 on top of stack

PUSHIMM 5
PUSHIMM 4
GREATER
STOP  //should leave 1 on top of stack
```



## SaM Simulator

1. What operations must SaM objects support?
2. How do we represent the internal state of SaM?
3. How do we load programs from a file?
4. How do we write code to interpret each of the opcodes?
5. What GUI do we use?

By the end of CS 211, you will be able to design and write moderately large, well-structured programs to simulate such systems.