What You’ve Learned (A Lot!)

Programming.
- Basic skills are universal (C, Java, PostScript, Maple, Perl, TeX).
- Key abstractions:
  - structured programming: for, while, if, function call
  - data structures: array, struct, linked list, stack, queue, tree
  - concepts: ADT, pointer, recursion, divide-and-conquer

- Can address important problems without relying on pre-packaged solutions.

COS 217
COS 226

What You’ve Learned (A Lot!)

Programming.
The TOY machine.
- Bridge between C language and hardware.
- Machine language programming (0’s and 1’s).
- von Neumann architecture.
- Building a TOY machine from logic gates.

COS 306

What You’ve Learned (A Lot!)

Programming.
The TOY machine.
Theory of computation.
- Use Turing machines to study computation.
- Computability: all machines have limitations.
- Church-Turing thesis: Turing machine is all-powerful.
- Algorithms: polynomial vs. exponential.
- Problem classes: P, NP, NP-complete.

COS 423
COS 487
What You’ve Learned (A Lot!)

Programming.
The TOY machine.
Theory of computation.
Odds and Ends.
  * Public-key cryptography.
  * Artificial Intelligence.

What Is Computer Science?

What is computer science?
  * The study of computation.

What is computation?
  * The process of manipulating and transforming information.

Why we learn CS.
  * Appreciate underlying principles.
  * Understand fundamental limitations.

An example: Lecture II: LFBSR TOY machine → ????
  * How to make a simple machine.
  * What can we do with it? What can't we do with it?
  * How fast can we do it?
  * Science behind it.

Course Themes

Layers of Abstraction.
  * Building a computer program.
    - divide program into small independent functions
    - ADT
  * Building a computer.
    - transistors → gates → maj, odd → adder → ALU
    - ALU, register file, decoder, multiplexer → TOY machine
  * Models of computation.
    - Turing machines, complexity classes

Course Themes

Layers of Abstraction.
Tradeoffs.
  * Time vs. space.
    - arrays, linked lists, BST
  * Program generality vs. simplicity.
  * Correct answer vs. time.
    - TSP brute force vs. heuristics
  * NP-completeness
  * New machine vs. new idea.
    - machine cost $ and makes “everything” run incrementally faster
    - new ideas can enable new research and technology
Course Themes

Layers of Abstraction.
Tradeoffs.
Self-reference.
- Recursion.
  - function that calls itself
- Linked list, tree.
  - self-referential data structures
- Fractal.
  - Mandelbrot set, H-tree pattern
- Sequential circuit.
  - feedback loop
- von Neumann architecture.
  - data and instruction stored in same main memory
- Universal Turing machine.
  - can simulate any machine including itself
- Undecidable problem.
  - key step in Halting proof was feeding one program itself as input

Course Themes

Layers of Abstraction.
Tradeoffs.
Self-reference.
- Re-use.
  - Loop.
    - let computer repeat code
  - Function.
    - re-use code
  - ADT.
    - build general purpose libraries
  - Circuit.
    - re-use primitive components
  - Divide-and-conquer.
    - re-use ideas recursively
  - Dynamic programming.
    - re-use results of intermediate computations

Programming Assignments

Assignments.
0. Hello world
1. Random numbers
2. Mandelbrot set
3. Rational arithmetic
4. Recursive graphics
5. TOY programming
6. Traveling salesperson
7. Data compression
8. Genetic code
9. & 10. RSA cryptosystem

"31809247817346517^{2837} \text{mod} 26381"
What To Do When You Face a New Problem?

What primitive objects are important?
- Numbers, files, pictures, text, programs, strings, matrices?
- Could always do it in C.
- Does another tool allow direct manipulation?

How long will it take me to do this task?
- Depends on what tool I use.

Have I done something like this before?
- If so, maybe I should use the same tool.
- Maybe I have some code laying around.
- Does it still work?

Will I be doing something like this again?
- If not, quick hack may be OK.

Will I be doing something like this *frequently*?
- Is it worthwhile to learn a new tool?
- Is it worthwhile to "create" a new tool?

Has "someone else" done something like this?
- May be some code laying around to reuse.

Will someone else be doing something like this in the future?
- Document the code?
- Make it portable?

"Whenever we think a problem is simple, it turns out to be complicated. Fortunately, whenever we think it to be complicated, it turns out to be simple."

---

Tips on Preparing for the Final

Final is comprehensive: covers the whole course.

Material since second midterm will be covered in greater depth.
- Theory.
  - Turing machines: tracing through and deducing purpose
  - Computability: basic ideas and significance
  - Analysis of algorithms (given code, predict how long it will take to solve problem)
  - P, NP, NP-complete, P = NP: basic ideas and definitions
- Crypto.
  - Basic ideas and definitions
  - Understand examples

---

Final Exam

Final.
- Friend Center Room 101.

Reading period office hours.
- To be posted on Web.

Rules.
- No computational devices.
- Closed note, closed book.
- Exception: one 8.5 x 11 page (both sides) in your own handwriting.
Where to go from Here?

- COS 217: Intro to Programming System.
- COS 226: Algorithms and Data Structures.
- COS 306: Logic Design. (a.k.a. ELE 206)