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
  - pointer, recursion, divide-and-conquer
- Can address important problems without relying on pre-packaged solutions.

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.

What You’ve Learned (A Lot!)

Programming.
The TOY machine.
Theory of computation.
- Use formal language to model computation.
- Use abstract machines to strip away inessential details.
- Computability: all machines have limitations.
- Church-Turing thesis: Turing machine is all-powerful.
- Algorithms: polynomial vs. exponential.
- Problem classes: P, NP, NP-complete.
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 I: LFDSR 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
- Formal languages.
  - abstraction to describe computation
- Models of computation.
  - abstract machines, complexity classes

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
- Expressiveness of language vs. ability to compile.
  - English is expressive: difficult for a computer to parse
  - C uses context-free grammar: easy to parse

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

M.C. Escher
© Cordon Art, Baam, the Netherlands
Course Themes

Reuse (don't reinvent the wheel):
  - Loop.
    - let computer repeat code
  - Program.
    - borrow similar program as template
  - Function.
    - reuse code
  - Circuit.
    - reuse primitive components
  - Divide-and-conquer.
    - reuse ideas recursively
  - ADT.
    - build general purpose libraries

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.

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)