CS 50
Introduction to Computer Science I

Michael D. Smith
smith@eecs.harvard.edu
Fall 2005

Q1: What's in common?

- Internet commerce and electronic markets
- Blockbuster movies and their special effects
- Medical research and life-support systems
- The space program
- The photocopier that made your handouts
- The car that brought me here today
- The iPod you listened to on the way to class

Each relies heavily on software and advances in computer science.

History of Computation

Calculi

- unknown origins

CS50 3

History of Computation

Calculi

- unknown origins

The abacus

- from 2600 BCE

CS50 4

History of Computation

Calculi
- unknown origins

The abacus
- from 2600 BCE

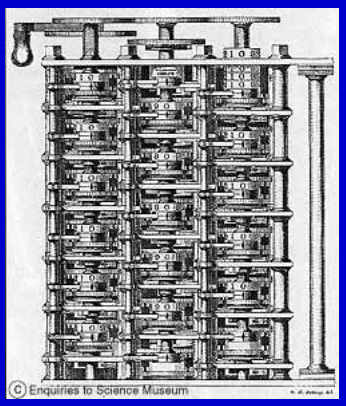
Calculating machines
- Napier's Bones, c. 1620

History of Computation

Calculi
- unknown origins

The abacus
- from 2600 BCE

Calculating machines
- Babbage and Byron,early 1800's



History of Computation

Calculi
- unknown origins

The abacus
- from 2600 BCE

Calculating machines
- Babbage and Byron,early 1800's

Theory of computation
- Turing, 1936

CS50 7

History of Computation

Calculi
- unknown origins

The abacus
- from 2600 BCE

Calculating machines
- Babbage and Byron,early 1800's

Theory of computation
- Turing, 1936

Programmable computers
- Aiken, 1944

CS50 8

History of Computation

Calculi
- unknown origins

The abacus
- from 2600 BCE

Calculating machines
- Babbage and Byron,early 1800's

Theory of computation
- Turing, 1936

Programmable computers
- Aiken, 1944
- ENIAC, 1946

History of Computation

Calculi
- unknown origins

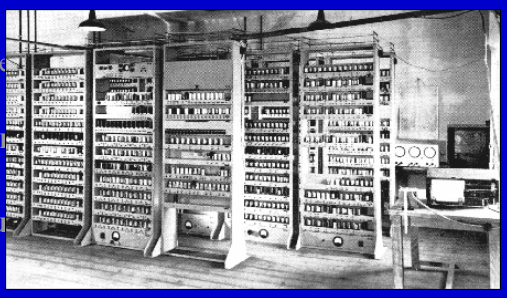
The abacus
- from 2600 BCE

Calculating machines
- Babbage and Byron,c 1800's

Theory of computatio
- Turing, 1936

Programmable compu
- Aiken, 1944
- ENIAC, 1946

Stored program computers
- Von Neumann, 1945
- EDSAC, 1949



Algorithms

Computer science began as the study of *mechanical processes*, now called *algorithms*

Example: Let's design an algorithm that tells us what else we might choose to do (academically) at 10am on MWF

A Computable Algorithm

A strategy for solving a problem that is

- **Precise**: clearly and unambiguously defined
- **Effective**: each step is capable of being executed
- **Finite**: expressed in bounded space and executable in bounded time

Is English a good language for expressing algorithms?

Programming Languages

Help humans to express algorithms precisely and effectively (only 2 of 3)

History

1940's: machine language
1950's: assembly language
1960's: imperative languages (e.g. Fortran, Algol, Cobol)
1970's: system-programming languages (e.g. C)
1980's: strongly-typed languages (e.g. Modula, ML)
object-oriented languages (e.g. Smalltalk, C++)
1990's: strongly-hyped† languages (e.g. Visual ..., Java)
2000's: scripting languages (e.g. Perl, Python, Ruby)

† *as stated by Kernighan*

CS50 13

Programming

programming:computer science
::
drafting:architecture

Drafting is

- the technical *lingua franca* of architecture
- a necessary background skill
- not coextensive with architecture
- not performed well by many excellent architects

CS50 14

Software Construction

- **"Coding" and "programming"**
  - are misleading terms
  - imply a mechanical (and boring) translation of preexisting designs into a computer language
  - and maybe all you did in high school CS
- **Software construction as we'll study it**
  - isn't mechanical
  - involves substantial creativity and judgment

Q2: What's the field?

"In what field can you walk into a sterile room, carefully controlled at 68°F, and find viruses, Trojan horses, worms, bugs, bombs, crashes, flames, twisted sex changers, and fatal errors?"

— from *Code Complete* by Steve McConnell

We'll program primarily in C

- **Invented in 1973 by Dennis Ritchie '63, PhD '68**
- **Preceded by**
  - B (Ken Thompson, Bell Labs, 1970)
  - BCPL (Martin Richards, Cambridge, 1967)
- **Designed for systems programming (operating systems, compilers)**
- **Not designed for pedagogy**

Why C?

Good things about the language:

- expressive, efficient, concise (cryptic?)
- permissive (doesn't get in your way)
- popular
- portable
- small step to other imperative language

Fits with my teaching style:

- bottom-up approach
- abstraction without the mystery

We'll also use Ruby

- **Invented in 1993 by Yukihiro "Matz" Matsumoto**
- **Scripting language**
  - more powerful than Perl
  - more object-oriented than Python
  - strong connections to Smalltalk
- **Designed to adhere to the Principle of Least Surprise**
  - makes programming fun
- **Ruby is written in C**

http://www.leuf.net/ww/wikidn?Jaoo2003Report

What you're missing

```
require 'net/http'

URL = 'HREF="http://www.registrar.fas.harvard.edu(/Courses/.*\.html)"'
TITLE = '">([-A-Za-z ]+ \d*\w*\. .*)<.A>'
TIME = '\(fall term\). M., W., F., at 10'

h = Net::HTTP.new('www.registrar.fas.harvard.edu', 80)
resp = h.get('/Courses/index.html', nil)
if resp.message == "OK"
   resp.body.scan(/#{URL}/) do
      resp2 = h.get($1, nil)
      if resp2.message == "OK"
         course = "NONE YET"
         resp2.body.each_line do |line|
            line.scan(/#{TITLE}/) { course = $1 }
            line.scan(/#{TIME}/) { puts "\t" + course }
         end
      end
   end
end
```

What is Computer Science 50?

Mathematics: What is computation and what is computable?

Science: How can these be computed? By what algorithms? What are the properties of these algorithms, including correctness and efficiency?

Engineering: How can actual systems that enable or manifest these computations be built and improved?

Course Coverage

Programming

- in C and Ruby
- under UNIX
- with associated tools

Basic concepts in computer science

- algorithms and analysis
- data representation and data types
- computer architecture

Design and implementation of algorithms

Additional topics

- networks
- security and cryptography
- theoretical foundations of computation
- artificial intelligence

Administration

http://
www.fas.harvard.edu/
~lib50

Prerequisites: none

Texts:

- Harel, *Computers Ltd.*
- Kernighan & Ritchie, *The C Programming Language*
- Schwartz, *Introduction to UNIX*
- [ONLINE] Thomas & Hunt, *Programming Ruby*
- [OPTIONAL] Roberts, *The Art and Science of C*

Teaching Fellows: excellent

Course Work

8 problem sets

- Roughly weekly
- Simple programs → An instant message client
- Typically 10-15 hours per week
- Collaboration policy
- Late policy

2 hourly exams and a final project

Sections and topic sessions