Overview

What is COS 126? Broad, but technical, intro to CS.

Goals.
- Demystify computer systems.
- Empower you to exploit available technology.
- Build awareness of substantial intellectual underpinnings.

Topics.
- Programming in Java.
- Machine architecture.
- Theory of computation.
- Applications to science, engineering, and commercial computing.

The Usual Suspects

Lectures. [Doug Clark]
- Tuesdays and Thursdays, Friend 101.

Precepts. [David Blei · Will Clarkson · Tom Funkhouser · Donna Gabai · Maia Ginsburg · Michael Golightly · Wyatt Lloyd · Anna Pop · Sid Sen · Jeff Terrace · Tao Yue]
- COS 126: Tue+Thu or Wed+Fri, various times (and places).
- ISC/CHM/COS/MOL/PHY 231/2: Wed or Th, 1:30, Icahn Lab 200.
  - (Wed people please follow instructions from HQ this week)
  - Tips on assignments, worked examples, clarify lecture material.

Friend 016/017 lab. [Undergrad lab assistants]
- Weekdays 7-11pm, some weekend hours.
- Full schedule on Web.

For full details: See www.princeton.edu/~cos126
Grades

*Course grades.* No preset curve or quota.

- **9 programming assignments.** 40%.
- **2 exams.** 50%.
- **Final programming project.** 10%.
- **Extra credit and staff discretion.** Adjust borderline cases.

Extra credit and staff discretion. Adjust borderline cases.

Course Materials

*Course website.* [www.princeton.edu/~cos126](http://www.princeton.edu/~cos126)

- Submit assignments, check grades.
- Programming assignments.
- Lecture notes.
  - Print slides before lecture; annotate during lecture.
  - At least skim before lecture; read thoroughly afterwards.

*Required readings.* Sedgewick and Wayne. *Intro to Programming in Java: An Interdisciplinary Approach.* [Labyrinth Books]

- Sedgewick and Wayne. *Intro to Programming in Java: An Interdisciplinary Approach.* [Labyrinth Books]

*Recommended.* Harel. *Computers Ltd.: What they really can’t do.* [Labyrinth]

Programming Assignments

*Desiderata.*

- Address an important scientific or commercial problem.
- Illustrate the importance of a fundamental CS concept.

*Examples.*

- N-body simulation.
- Pluck a guitar string.
- DNA sequence alignment.
- Estimate Avogadro’s number.

Due. **(Usually) Mondays 11 pm via Web submission.**

Computing equipment.

- Your laptop. [OS X, Windows, Linux, iPhone, ...]
- OIT desktop. [Friend 016 and 017 labs]

Survival Guide

*Keep up with the course material.*

- Attend lectures and precepts.
- Take notes.
  - At least skim readings before lecture; read more closely after.
- Plan multiple sessions for programming assignments.
- Visit course home page regularly.

*Ask for help when you need it!*

- Preceptors / instructor.
  - Email, office hours, precepts
  - Concepts, programming assignments
- Undergrad Lab TAs.
  - OS support, help with minor debugging
What’s Ahead?

Tuesday. Lecture 2: Intro to Java.

*COS 126 Precept 1. Meets today or tomorrow.
ICS/CHM/COS/MOL/PHY 231/2. Meets today and/or tomorrow

Not registered? Go to any precept today; officially register ASAP.
Change precepts? Use SCORE.

ASAP! Read sections 1.1 - 1.2 of Intro to Programming.

Assignment 0. Due Monday 9/15, 11 pm.

- Install Java programming environment + a few exercises.
- Lots of help available, don’t be bashful.

END OF ADMINISTRATIVE STUFF

Prologue: A Simple Machine

Secure Chat

Alice wants to send a secret message to Bob?

- Can you read the secret message $gX76W3v7K$?
- But Bob can. How?

Encryption Machine

Goal. Design a machine to encrypt and decrypt data.

<table>
<thead>
<tr>
<th>SEND MONKEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>encrypt</td>
</tr>
<tr>
<td>gX76W3v7K</td>
</tr>
<tr>
<td>decrypt</td>
</tr>
<tr>
<td>SEND MONKEY</td>
</tr>
</tbody>
</table>
Encryption Machine

**Goal.** Design a machine to encrypt and decrypt data.

```
SENDMONEY
```

encrypt

```
gX76W3v7k
```

decrypt

```
SENDMONEY
```

**Enigma encryption machine.**
- “Unbreakable” German code during WWII.
- Broken by Turing bombe.
- One of first uses of computers.
- Helped win Battle of Atlantic by locating U-boats.

---

A Digital World

**Data is a sequence of bits.**
- Text.
- Documents, pictures, sounds, movies, ...
- Programs, executables.

**File formats.** txt, pdf, doc, ppt, jpeg, mp3, divx, java, exe, ...

---

Base64 encoding. Use 6 bits to represent each alphanumeric symbol.

<table>
<thead>
<tr>
<th>Binary Char</th>
<th>Binary Char</th>
<th>Binary Char</th>
<th>Binary Char</th>
<th>Binary Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000 A</td>
<td>001011 L</td>
<td>010101 W</td>
<td>100001 h</td>
<td>101100 s</td>
</tr>
<tr>
<td>000001 B</td>
<td>001100 M</td>
<td>010111 X</td>
<td>100100 i</td>
<td>101101 t</td>
</tr>
<tr>
<td>000010 C</td>
<td>001101 N</td>
<td>010000 Y</td>
<td>100011 j</td>
<td>101110 w</td>
</tr>
<tr>
<td>000011 D</td>
<td>001110 O</td>
<td>010001 Z</td>
<td>100100 k</td>
<td>101111 v</td>
</tr>
<tr>
<td>000100 E</td>
<td>001111 P</td>
<td>011010 a</td>
<td>101001 l</td>
<td>110000 w</td>
</tr>
<tr>
<td>000101 F</td>
<td>010000 Q</td>
<td>011011 b</td>
<td>100110 m</td>
<td>110100 x</td>
</tr>
<tr>
<td>000110 G</td>
<td>010001 R</td>
<td>011100 c</td>
<td>100111 n</td>
<td>110110 y</td>
</tr>
<tr>
<td>000111 H</td>
<td>010010 S</td>
<td>011101 d</td>
<td>101000 o</td>
<td>110111 z</td>
</tr>
<tr>
<td>001000 I</td>
<td>010011 T</td>
<td>011110 e</td>
<td>101001 p</td>
<td>110100 0</td>
</tr>
<tr>
<td>001001 J</td>
<td>010100 U</td>
<td>011111 f</td>
<td>101010 q</td>
<td>110101 1</td>
</tr>
<tr>
<td>001010 K</td>
<td>010101 V</td>
<td>100000 g</td>
<td>101011 r</td>
<td>110110 2</td>
</tr>
</tbody>
</table>
One-Time Pad Encryption

Encryption.
- Convert text message to N bits.
- Generate N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.
- Convert binary back into text.

Base64 Encoding

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Y</td>
<td>24</td>
<td>011000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

message

SENDMDONEY

base64

010010 000100 001101 000011 001100 001110 001101 000010 011001

random bits

110010 010011 110110 111001 011010 111001 100010 111111 010010

XOR

100000 010111 110110 111001 011010 111001 100010 111111 010010

Encrypted

010010 000100 001101 000011 001100 001110 001101 000010 011001

XOR Truth Table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x \oplus y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

sum corresponding pair of bits: 1 if sum is odd, 0 if even
Secure Chat

Alice wants to send a secret message to Bob?

- Can you read the secret message \( gX76W3v7K \) ?

But Bob can. How?

One-Time Pad Decryption

Decryption.

- Convert encrypted message to binary.

```
g X 7 6 W 3 v 7 K
100000 010111 111011 111010 010110 110111 110111 001010
```

Base64 Encoding

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>23</td>
<td>010111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

One-Time Pad Decryption

Decryption.

- Convert encrypted message to binary.

```
g X 7 6 W 3 v 7 K
100000 010111 111011 111010 010110 110111 110111 001010
```

- Use same \( N \) random bits (one-time pad).
One-Time Pad Decryption

Decryption.
- Convert encrypted message to binary.
- Use same N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X ^ Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Why Does It Work?

Crucial property. Decrypted message = original message.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>original message bit</td>
</tr>
<tr>
<td>b</td>
<td>one-time pad bit</td>
</tr>
<tr>
<td>^</td>
<td>XOR operator</td>
</tr>
<tr>
<td>a ^ b</td>
<td>encrypted message bit</td>
</tr>
<tr>
<td>(a ^ b) ^ b</td>
<td>decrypted message bit</td>
</tr>
</tbody>
</table>

Why is crucial property true?
- Use properties of XOR.
- \((a ^ b) ^ b = a ^ (b ^ b) = a ^ 0 = a\)

XOR Truth Table

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X ^ Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Goods and Bads of One-Time Pads

Good.
- Very simple encryption/decryption processes.
- Provably unbreakable if pad is truly random. [Shannon, 1940s]

Bad.
- Easily breakable if pad is re-used.
- Pad must be as long as the message.
- Truly random bits are very hard to come by.
- Pad must be distributed securely.

Eavesdropper Eve sees only random bits. "one time" means one time only.

impractical for Web commerce
Random Numbers

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

Jon von Neumann (left), ENIAC (right)

Pseudo-Random Bit Generator

Practical middle-ground.
- Let’s make a pseudo-random bit generator gadget.
- Alice and Bob each get identical small gadgets.

How to make small gadget that produces "random" numbers.
- Linear feedback shift register.
- Linear congruential generator.
- Blum-Blum-Shub generator.
- ...

Shift Register

Shift register terminology.
- Bit: 0 or 1.
- Cell: storage element that holds one bit.
- Register: sequence of cells.
- Seed: initial sequence of bits.
- Shift register: when clock ticks, bits propagate one position to left.

Linear Feedback Shift Register

\((8, 10)\) linear feedback shift register.
- 11 bit shift register.
- New output bit 0 is XOR of previous bits 8 and 10.
- Output bit = bit 0.

LFSR demo

\[ \begin{array}{ccccccccccc}
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array} \]

time \( t \)

\[ \begin{array}{ccccccccccc}
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & ? \\
\end{array} \]

time \( t + 1 \)
Random Numbers

Q. Are these 2000 numbers random? If not, what is the pattern?

A. No. This is output of an 11 bit LFSR!

The Science Behind A LFSR

Q. Are the bits really random?
A. No! Real machines are deterministic.

Q. Will bit pattern repeat itself?
A. Yes, after $2^{11} - 1 = 2047$ steps.

Q. What if I need more bits?
A. Scalable: 20 cells for 1 million bits, 30 for 1 billion.

Q. Will the machine work equally well if we XOR bits 4 and 10?
A. No! Need to understand theory of finite groups.

Q. How many cells do I need to guarantee a certain level of security?
A. Subject of active research.

Other LFSR Applications

What else can we do with a LFSR?
- DVD encryption with CSS.
- DVD decryption with DeCSS!
- Subroutine in military cryptosystems.

LFSR and "General Purpose Computer"

Important properties.
- Built from simple components.
- Scales to handle huge problems.
- Requires a deep understanding to use effectively.

Critical difference. General purpose machine can be programmed to simulate ANY abstract machine.
A Java program.
- Prints same bits as LFSR.
- The code will make sense next week!

```java
public class LFSR {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        boolean b10 = false, b9 = true, b8 = true, b7 = false;
        boolean b6 = true, b5 = false, b4 = false, b3 = false;
        boolean b2 = false, b1 = true, b0 = false;

        for (int i = 0; i < N; i++) {
            boolean bit = b8 ^ b10;
            b10 = b9;
            b9 = b8;
            b8 = b7;
            b7 = b6;
            b6 = b5;
            b5 = b4;
            b4 = b3;
            b3 = b2;
            b2 = b1;
            b1 = b0;
            b0 = bit;

            if (bit) System.out.print(1);
            else System.out.print(0);
        }
    }
}
```

% java LFSR 2000
11011001001111011011100101101011100110011010011011011
1011101110001111111111111111010010000100111010011110110010111