Overview

What is COS/EGR 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.

“Computers are incredibly fast, accurate, and stupid; humans are incredibly slow, inaccurate, and brilliant; together they are powerful beyond imagination.” – Albert Einstein

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.

participation helps, frequent absences hurts
Course Materials

Course website. [www.princeton.edu/~cos126]
- Submit assignments, check grades.
- Programming assignments.
- Lecture slides.

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

Recommended readings. Harel. *What computers can't do.* [Labyrinth]

Princeton royalties donated to ACM-W this semester

Programming Assignments

Desiderata.
- Address an important scientific or commercial problem.
- Illustrate the importance of a fundamental CS concept.
- You solve problem from scratch!

Due. Mondays 11pm via Web submission.

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

What’s Ahead?

Lecture 2. Intro to Java.

Precept 1. Meets today/tomorrow.
Precept 2. Meets Thu/Fri.

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

Assignment 0. Due Monday, 11pm.
- Read Sections 1.1 and 1.2 in textbook.
- Install Java programming environment + a few exercises.
- Lots of help available, don’t be bashful.

END OF ADMINISTRATIVE STUFF
0. 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.

![Encryption Machine](image)

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.
Data is a sequence of bits. [bit = 0 or 1]

- Text.
- Programs, executables.
- Documents, pictures, sounds, movies, ...

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

```
<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>010110 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>100010 i</td>
<td>101101 t</td>
</tr>
<tr>
<td>000100 C</td>
<td>001101 N</td>
<td>011000 Y</td>
<td>100101 j</td>
<td>101110 u</td>
</tr>
<tr>
<td>001000 D</td>
<td>011010 O</td>
<td>011001 Z</td>
<td>10100 k</td>
<td>10111 v</td>
</tr>
<tr>
<td>001001 E</td>
<td>011111 P</td>
<td>011010 a</td>
<td>10101 l</td>
<td>110000 w</td>
</tr>
<tr>
<td>001010 F</td>
<td>010000 Q</td>
<td>011011 b</td>
<td>100110 m</td>
<td>110001 x</td>
</tr>
<tr>
<td>001100 G</td>
<td>010001 R</td>
<td>011100 c</td>
<td>10111 n</td>
<td>11010 y</td>
</tr>
<tr>
<td>001111 H</td>
<td>010010 S</td>
<td>011101 d</td>
<td>101000 o</td>
<td>11011 z</td>
</tr>
<tr>
<td>010001 J</td>
<td>010011 T</td>
<td>011110 e</td>
<td>101001 p</td>
<td>110100 0</td>
</tr>
<tr>
<td>010100 K</td>
<td>010100 U</td>
<td>011111 f</td>
<td>101010 q</td>
<td>110111 1</td>
</tr>
<tr>
<td>010101 L</td>
<td>010101 V</td>
<td>100000 g</td>
<td>101111 r</td>
<td>110110 2</td>
</tr>
</tbody>
</table>
```

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

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>N</th>
<th>E</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td>001000</td>
<td>001101</td>
<td>000011</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
<td></td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>110001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010100</td>
<td>110111</td>
<td>111011</td>
<td>001010</td>
<td></td>
</tr>
</tbody>
</table>

message
base64
random bits
XOR

0 ^ 1 = 1

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

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>
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.

<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>W</td>
<td>22</td>
<td>010110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Base64 Encoding

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>w</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>

encrypted

base64

One-Time Pad Decryption

Decryption.
- Convert encrypted message to binary.
- Use same N random bits (one-time pad).

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>w</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>110001</td>
<td>011010</td>
<td>111011</td>
<td>101111</td>
<td>111111</td>
<td>010010</td>
</tr>
</tbody>
</table>

encrypted

base64

random bits
One-Time Pad Decryption

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

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>

Why Does It Work?

Crucial property. Decrypted message = original message.

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

Why Does It Work? (with the wrong pad)

Decryption.
- Convert encrypted message to binary.
- Take bitwise XOR of two bitstrings.
- Convert back into text.
One-Time Pad Decryption (with the wrong pad)

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

<table>
<thead>
<tr>
<th>g</th>
<th>x</th>
<th>7</th>
<th>6</th>
<th>w</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>010000</td>
<td>110111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>111011</td>
<td>001010</td>
<td></td>
</tr>
<tr>
<td>010100</td>
<td>111010</td>
<td>111011</td>
<td>110101</td>
<td>010010</td>
<td>110101</td>
<td>101010</td>
<td>001010</td>
<td></td>
</tr>
<tr>
<td>001000</td>
<td>001011</td>
<td>001110</td>
<td>010101</td>
<td>000100</td>
<td>001110</td>
<td>001010</td>
<td>010001</td>
<td>000000</td>
</tr>
</tbody>
</table>

XOR

base64

<table>
<thead>
<tr>
<th>g</th>
<th>x</th>
<th>7</th>
<th>6</th>
<th>w</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>010000</td>
<td>110111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>111011</td>
<td>001010</td>
<td></td>
</tr>
<tr>
<td>101000</td>
<td>011010</td>
<td>110101</td>
<td>111101</td>
<td>010101</td>
<td>111011</td>
<td>100101</td>
<td>101010</td>
<td>001010</td>
</tr>
</tbody>
</table>

I L O V E O K R A

Wrong message
Goods and Bads of One-Time Pads

Good.
- Easily computed by hand.
- Very simple encryption/decryption processes.
- Provably unbreakable if bits are 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.

“one time” means one time only
impractical for Web commerce

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 pseudo-random numbers.
- Linear feedback shift register.
- Linear congruential generator.
- Blum-Blum-Shub generator.
- ...

“Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.”
—Jon von Neumann (left)
—ENIAC (right)

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 (LFSR)

(8, 10) linear feedback shift register.
- Shift register with 11 cells.
- Bit $b_0$ is XOR of previous bits $b_8$ and $b_{10}$.
- Pseudo-random bit $= b_0$.

LFSR Encryption
- Convert text message to N bits.
- Initialize LFSR with small seed.
- Generate $N$ random bits with LFSR.
- Take bitwise XOR of two bit-strings.
- Convert binary back into text.

LFSR Decryption
- Convert encrypted message to binary.
- Initialize identical LFSR with same small seed.
- Generate $N$ random bits with LFSR.
- Take bitwise XOR of two bit-strings.
- Convert back into text.

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

A. No. This is output of (8, 10) LFSR with seed 01101000010!
Goods and Bads of LFSR Encryption

Goods.
- Easily computed with simple machine.
- Very simple encryption / decryption process.
- Scalable: 20 cells for 1 million bits; 30 cells for 1 billion bits.
  [but need theory of finite groups to know where to put taps]

Bads.
- Still need secure, independent way to distribute LFSR seed.
- The bits are not truly random.
  [bits in our 11-bit LFSR cycle after $2^{11} - 1 = 2047$ steps]
- Experts have cracked LFSR.
  [more complicated machines needed]

LFSR and "General Purpose Computer"

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

<table>
<thead>
<tr>
<th>Basic Component</th>
<th>LFSR</th>
<th>Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>control</td>
<td>start, stop, load</td>
<td>same</td>
</tr>
<tr>
<td>clock</td>
<td>regular pulse</td>
<td>2.8 GHz pulse</td>
</tr>
<tr>
<td>memory</td>
<td>11 bits</td>
<td>1 GB</td>
</tr>
<tr>
<td>input</td>
<td>seed</td>
<td>sequence of bits</td>
</tr>
<tr>
<td>computation</td>
<td>shift, XOR</td>
<td>logic, arithmetic, ...</td>
</tr>
<tr>
<td>output</td>
<td>pseudo-random bits</td>
<td>Sequence of bits</td>
</tr>
</tbody>
</table>

Critical difference. General purpose machine can be programmed to simulate ANY abstract machine.

Programming. Can write a Java program to simulate the operations of any abstract machine.
- Basis for theoretical understanding of computation. [stay tuned]
- Basis for bootstrapping real machines into existence. [stay tuned]

Stay tuned. See Assignment 5.

Other LFSR Applications

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

A Profound Idea

public class LFSR {
    private int seed[];
    private int tap;
    private int N;
    public LFSR(String seed, int tap) { ... }
    public int step() { ... }
    public static void main(String[] args) {
        LFSR lfsr = new LFSR("01101000010", 8);
        for (int i = 0; i < 2000; i++)
            StdOut.println(lfsr.step());
    }
}
A Profound Question

Q. What is a random number?

LFSR does not produce random numbers.
- It is a very simple deterministic machine.
- But not obvious how to distinguish the bits it produces from random.

Q. Are random processes found in nature?
- Motion of cosmic rays or subatomic particles?
- Mutations in DNA?

Q. Is the natural world a (not-so-simple) deterministic machine?

“God does not play dice.”

— Albert Einstein