Overview

What is COS 126?
- Broad, but technical, intro to CS in the context of scientific, engineering, and commercial applications.
- No prerequisites, intended for novices.

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

Topics.
- Programming in Java.
- Machine architecture.
- Theory of computation.
- Applications.

The Usual Suspects

Lectures. [Doug Clark + Kevin Wayne]
- Tuesdays and Thursdays, Frist 302.
- Same lecture and lecturer at 10 and 11.

- Tue+Thu or Wed+Fri.
- Tips on assignments, worked examples, clarify lecture material.

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

Grades

Course grades. No preset curve or quota.

Programming assignments. 35%.
2 exams. 50%.
Final project. 15%.

Staff discretion. Adjust borderline cases.

you are here

can drop lowest one

you are here

y0u a re he r e
Course Materials

Course website. [www.princeton.edu/~cos126]
- Programming assignments.
- Lecture notes.
- Exam archive.

Required readings.
- Sedgewick and Wayne. Intro to Computer Science.
  - draft of some sections [Triangle copy, 150 Nassau]
  - booksite [www.cs.princeton.edu/IntroCS]

Recommended readings.
- Harel. What computers can't do. [U-Store]

Programming Assignments

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

Examples.
- N-body simulation.
- DNA sequence alignment.
- Digital signal processing of MP3 files.
- Traveling salesperson problem.
- Markov model of natural language.

Due. Mondays 11:55pm via Web submission.

Computing equipment.
- Your machine. [Linux, OS X, Windows, Java cell phone, ... ]
- OIT machines. [Friend 016 and 017 labs]

What’s Ahead?

First precept. Meets today/tomorrow.
- Don’t know yours? Check the print out or website.
- See Donna Gabai after class if not registered.

Thursday lecture. Intro to Java.

Assignment 0. Due Monday 2/13.
- Read Sections 1.1 and 1.2 in Intro to CS.
- Setup Java programming environment + a few exercises.
- Lots of help available, don’t be bashful.

END OF ADMINISTRATIVE STUFF
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.

![Enigma encryption machine.](image)

"Unbreakable" German code during WWII.
- Broken by Turing bombe.
- One of first uses of computers.
- Helped win Battle of Atlantic by locating U-boats.

Digital Data

Computers store all data as a sequence of bits.
- Text.
- Images (JPEG), audio (MP3), video (DivX).
- Programs.

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>010110 W</td>
<td>100001 h</td>
<td>101100 s</td>
</tr>
<tr>
<td>000000 B</td>
<td>001100 M</td>
<td>010111 X</td>
<td>100010 i</td>
<td>101101 t</td>
</tr>
<tr>
<td>000010 C</td>
<td>001111 N</td>
<td>011000 Y</td>
<td>100111 j</td>
<td>111001 u</td>
</tr>
<tr>
<td>000011 D</td>
<td>001110 O</td>
<td>011100 Z</td>
<td>101000 k</td>
<td>111011 v</td>
</tr>
<tr>
<td>000100 E</td>
<td>010011 P</td>
<td>011010 a</td>
<td>101001 l</td>
<td>111000 w</td>
</tr>
<tr>
<td>000101 F</td>
<td>010000 Q</td>
<td>011011 b</td>
<td>101010 m</td>
<td>111001 x</td>
</tr>
<tr>
<td>000110 G</td>
<td>010001 R</td>
<td>011100 c</td>
<td>101101 n</td>
<td>111010 y</td>
</tr>
<tr>
<td>000111 H</td>
<td>010010 S</td>
<td>011101 d</td>
<td>101100 o</td>
<td>111101 z</td>
</tr>
<tr>
<td>001000 I</td>
<td>010100 T</td>
<td>011110 e</td>
<td>101100 p</td>
<td>111110 0</td>
</tr>
<tr>
<td>001001 J</td>
<td>010100 U</td>
<td>011111 f</td>
<td>101101 q</td>
<td>111111 1</td>
</tr>
<tr>
<td>001010 K</td>
<td>010101 V</td>
<td>100000 g</td>
<td>101010 r</td>
<td>110110 2</td>
</tr>
</tbody>
</table>

One-Time Pad Encryption

Encryption.
- Convert text message to $N$ bits.

<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>

Base64 Encoding

message = $010010 001000 000001 000011 000110 001100 001101 001101 001110 001111 001100 010000$
One-Time Pad Decryption

Decryption.
- Convert encrypted message to binary.
- Use same N random bits (one-time pad).
- Take bitwise XOR of two strings.
- 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>g</td>
<td>23</td>
<td>010111</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 strings.
- Convert encrypted message to binary.

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>X</td>
<td>23</td>
<td>010111</td>
</tr>
</tbody>
</table>

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>

<table>
<thead>
<tr>
<th>SENDMONEY</th>
<th>message</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td>base64</td>
</tr>
<tr>
<td>000100</td>
<td></td>
</tr>
<tr>
<td>011010</td>
<td></td>
</tr>
<tr>
<td>001011</td>
<td></td>
</tr>
<tr>
<td>011101</td>
<td></td>
</tr>
<tr>
<td>001010</td>
<td></td>
</tr>
<tr>
<td>011000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<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>00000</td>
<td>010111</td>
<td>110111</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>

<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>110111</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>111011</td>
<td>001010</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
<th>encrypted</th>
</tr>
</thead>
<tbody>
<tr>
<td>110010</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>011010</td>
<td>111011</td>
<td>111011</td>
<td>010100</td>
<td>base64</td>
</tr>
<tr>
<td>110010</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>011010</td>
<td>111011</td>
<td>111011</td>
<td>010100</td>
<td>random bits</td>
</tr>
</tbody>
</table>
One-Time Pad Decryption

Why Does It Work?

- **Notation** | **Meaning**
  - a | original message
  - b | one-time pad
  - ^ | XOR operator
  - a ^ b | encrypted message
  - (a ^ b) ^ b | decrypted message

**Crucial property:** (a ^ b) ^ b = a.
- Decrypted message = original message.

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

Good.
- Very simple encryption/decryption processes.
- Theoretically unbreakable if pad is truly random.

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

Practical middle-ground.
- Let's make a pseudo-random bit generator gadget.
- Alice and Bob each get identical small gadgets instead of identical large one-time pads.

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

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

LFSR and “General Purpose Computer”

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

Other LFSR Applications

- DVD encryption with CSS.
- DVD decryption with DeCSS!
- Subroutine in military cryptosystems.

What else can we do with a LFSR?

Java program prints same bits as LFSR.
- You’ll understand this program by 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; // update

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

Reference: [http://www.csee.usf.edu/~dft/DeCSS/Gallery](http://www.csee.usf.edu/~dft/DeCSS/Gallery)