General Computer Science
Princeton University
Spring 2011

Kevin Wayne

www.princeton.edu/~cos126
Overview

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

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
The Basics

Lectures. [Kevin Wayne]
- Tuesdays and Thursdays, Frist 302.
- Same lecture at 10am and 11am.
- Office hours: M Th 2-3pm in CS 207.

Precepts. [Donna Gabai (co-lead) · Keith Vertanen (co-lead) · 11 others]
- Tue+Thu or Wed+Fri.
- Tips on assignments, worked examples, clarify lecture material.

Computing laboratory. [Undergrad lab assistants]
- Sun 5-11pm, Mon-Fri 7-11pm, Sat 2-6pm in Friend 017.
- Help with debugging.

Full details and office hours. 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.

participation helps, frequent absences hurts

Check grades. Blackboard. [www.blackboard.princeton.edu]
Course Materials

Course website. [www.princeton.edu/~cos126]
- Programming assignments and checklists.
- Submit assignments.
- Lecture slides. print before lecture; annotate during lecture
- Exam archive.

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

Princeton royalties donated to ACM-W this semester

Recommended readings. Harel. *What computers can’t do.* [Labyrinth]
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?
- Sometime in the past, they exchange a one-time pad.
- Alice uses the pad to encrypt the message.
- Bob uses the same pad to decrypt the message.

Key point. Without the pad, Eve cannot understand the message.
Encryption Machine

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

<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>g</td>
<td>X</td>
<td>7</td>
<td>6</td>
<td>W</td>
<td>3</td>
<td>v</td>
<td>7</td>
<td>K</td>
</tr>
</tbody>
</table>

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. \[ \text{bit} = 0 \text{ or } 1 \]
- Text.
- Programs, executables.
- Documents, pictures, sounds, movies, ...

File formats. txt, pdf, java, exe, docx, pptx, jpeg, mp3, divx, ...
A Digital World

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

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

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

<table>
<thead>
<tr>
<th>Binary</th>
<th>Char</th>
<th>Binary</th>
<th>Char</th>
<th>Binary</th>
<th>Char</th>
<th>Binary</th>
<th>Char</th>
<th>Binary</th>
<th>Char</th>
<th>Binary</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>A</td>
<td>001011</td>
<td>L</td>
<td>010110</td>
<td>W</td>
<td>100001</td>
<td>h</td>
<td>101100</td>
<td>s</td>
<td>110111</td>
<td>3</td>
</tr>
<tr>
<td>000001</td>
<td>B</td>
<td>001100</td>
<td>M</td>
<td>010111</td>
<td>X</td>
<td>100010</td>
<td>i</td>
<td>101101</td>
<td>t</td>
<td>111000</td>
<td>4</td>
</tr>
<tr>
<td>000010</td>
<td>C</td>
<td>001101</td>
<td>N</td>
<td>011000</td>
<td>Y</td>
<td>100011</td>
<td>j</td>
<td>101110</td>
<td>u</td>
<td>111001</td>
<td>5</td>
</tr>
<tr>
<td>000011</td>
<td>D</td>
<td>001110</td>
<td>O</td>
<td>011001</td>
<td>Z</td>
<td>101000</td>
<td>k</td>
<td>101111</td>
<td>v</td>
<td>111010</td>
<td>6</td>
</tr>
<tr>
<td>000100</td>
<td>E</td>
<td>001111</td>
<td>P</td>
<td>011010</td>
<td>a</td>
<td>101010</td>
<td>l</td>
<td>110000</td>
<td>w</td>
<td>111011</td>
<td>7</td>
</tr>
<tr>
<td>000101</td>
<td>F</td>
<td>010000</td>
<td>Q</td>
<td>011011</td>
<td>b</td>
<td>101110</td>
<td>m</td>
<td>110001</td>
<td>x</td>
<td>111100</td>
<td>8</td>
</tr>
<tr>
<td>000110</td>
<td>G</td>
<td>010001</td>
<td>R</td>
<td>011100</td>
<td>c</td>
<td>101111</td>
<td>n</td>
<td>110010</td>
<td>y</td>
<td>111110</td>
<td>9</td>
</tr>
<tr>
<td>000111</td>
<td>H</td>
<td>010010</td>
<td>S</td>
<td>011101</td>
<td>d</td>
<td>101000</td>
<td>o</td>
<td>110011</td>
<td>z</td>
<td>111110</td>
<td>+</td>
</tr>
<tr>
<td>001000</td>
<td>I</td>
<td>010011</td>
<td>T</td>
<td>011110</td>
<td>e</td>
<td>101001</td>
<td>p</td>
<td>110100</td>
<td>0</td>
<td>111111</td>
<td>/</td>
</tr>
<tr>
<td>001001</td>
<td>J</td>
<td>010100</td>
<td>U</td>
<td>011111</td>
<td>f</td>
<td>101010</td>
<td>q</td>
<td>110101</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001010</td>
<td>K</td>
<td>010101</td>
<td>V</td>
<td>100000</td>
<td>g</td>
<td>101011</td>
<td>r</td>
<td>110110</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
One-Time Pad Encryption

Encryption.
- Convert text message to N bits.

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>M</td>
<td>12</td>
<td>001100</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

S E N D M O N E Y

message

010010 000100 001101 000011 001100 001110 001101 000100 011000

base64
One-Time Pad Encryption

**Encryption.**
- Convert text message to N bits.
- Generate N random bits (one-time pad).

```
S E N D M O N E Y
010010 000100 001101 000011 001100 001110 001101 000100 011000
```

```
message
010010 010011 110110 111001 011010 111001 100010 111111 010010
```

```
random bits
110010 010011 110110 111001 011010 111001 100010 111111 010010
```

base64
One-Time Pad Encryption

Encryption.

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

**Encryption.**

<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>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>111001</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>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</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>

message

base64

random bits

XOR

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

\[ 0 ^ 1 = 1 \]
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>w</td>
<td>22</td>
<td>010110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<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>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>111001</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>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>

message
base64
random bits
XOR
encrypted

g  X  7  6  W  3  v  7  K
One-Time Pad Decryption

Decryption.
- Convert encrypted message to binary.
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>

Encrypted message:

```
g X 7 6 W 3 v 7 K
100000 010111 111011 111010 010110 110111 101111 111011 001010
```

Base64 Encoding
One-Time Pad Decryption

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

```
<table>
<thead>
<tr>
<th></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>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td>010111 111011 11010 010110</td>
<td>110111 101111 111011 001010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110010</td>
<td>010011 110110 11001 011010</td>
<td>111001 100010 111111 010010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>

<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>111001</td>
<td>011010</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
</tbody>
</table>

encrypted
base64
random bits
XOR

\[ 1 \oplus 1 = 0 \]
One-Time Pad Decryption

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

<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>110111</td>
<td>111001</td>
<td>011010</td>
<td>111001</td>
<td>000100</td>
<td>110111</td>
<td>010010</td>
</tr>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
</tbody>
</table>

encrypted
base64
random bits
XOR
message

<table>
<thead>
<tr>
<th>Base64 Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>M</td>
</tr>
<tr>
<td>...</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>
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.
- Convert back into text: **Oops**.

\[
\begin{array}{cccccccccc}
g & X & 7 & 6 & W & 3 & v & 7 & K & \\
100000 & 010111 & 111011 & 11010 & 010110 & 110111 & 101111 & 111011 & 001010 & \\
101000 & 011100 & 110101 & 101111 & 010010 & 111001 & 100101 & 101010 & 001010 & \\
001000 & 001011 & 001110 & 010101 & 000100 & 001110 & 001010 & 010001 & 000000 & \\
\end{array}
\]

encrypted

base64

**wrong bits**

**XOR**

**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.
Pseudo-Random Bit Generator

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

How to make small gadget that produces pseudo-random numbers.
- Enigma.
- 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 \( \text{XOR} \) of previous bits \( b_8 \) and \( b_{10} \).
- Pseudo-random bit = \( b_0 \).

![LFSR Diagram]
Q. Are these 2000 numbers random? If not, what is the pattern?

A. No. This is output of \{8, 10\} LFSR with seed 01101000010!
LFSR Encryption

LFSR encryption.
- Convert text message to N bits.
- Initialize LFSR with small seed.
- Generate N bits with LFSR.
- 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>w</td>
<td>22</td>
<td>010110</td>
</tr>
</tbody>
</table>

message

base64

LFSR bits

XOR

encrypted
LFSR Decryption

**LFSR Decryption.**

- Convert encrypted message to binary.
- Initialize identical LFSR with same seed.
- Generate N bits with LFSR.
- Take bitwise XOR of two bitstrings.
- Convert binary back into text.

<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>111001</td>
<td>011010</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
<tr>
<td>S</td>
<td>E</td>
<td>N</td>
<td>D</td>
<td>M</td>
<td>O</td>
<td>N</td>
<td>E</td>
<td>Y</td>
</tr>
</tbody>
</table>

**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>M</td>
<td>12</td>
<td>001100</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**LFSR bits**

**encrypted**

**message**

**XOR**
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 ]
Other LFSR Applications

What else can we do with a LFSR?

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

---

```c
/*     efdtt.c     Author:  Charles M. Hannum <root@ihack.net>             */
/*     Usage is:  cat title-key scrambled.vob | efdtt >clear.vob           */
#define m(i)(x[i]^s[i+84])<<

unsigned char x[5] , y, s[2048];
main(n){
for( read(0,x,5); read(0,s, n=2048)
    );
write(1, s,n)
    }

if(s [y=s [13]%8+20] /16%4 ==1){
int i=m(1)17 ^256 +m(0) 8,k =m(2)
0, j=  m(4) 17^ m(3) 9^k* 2-k%8
^8,a =0,c =26;for (s[y] -=16;
--c; j *=2)
a=a2^i& 1, i=i /2^j&1
<y) ++j<n;c=c>
c 
+=y=i^i/8^i>>4^i>>12,
i=i>>8^y<17,a^=a>>14,y=a^a*8^a<<6,a=a
>>8^y<9,k=s[j],k ="7Wo~'G_\216"[k &7]+2^"cr3sfw6v:*k+>/n."[k>>4]*2^k*257/
8,s[j]=k^(k&k+2&34)*6^c+~y
;
}
```

http://www.cs.cmu.edu/~dst/DeCSS/Gallery
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.
A Profound Idea

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.

```java
public class LFSR {
    private int seed[];
    private final int tap;
    private final 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());
    }
}

% java LFSR
11001001001111011011100101101
01110011000101111110100100001
00110100101111001100100111...
```
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. Or, is the natural world a (not-so-simple) deterministic machine?

“God does not play dice.”
— Albert Einstein