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.

"Computers are incredibly fast, accurate, and stupid; Humans are incredibly slow, inaccurate, and brilliant; together they are powerful beyond imagination."

The Basics

Lectures. [Kevin Wayne]
- Tuesdays and Thursdays, Frist 302.
- Same lecture at 10am and 11am.

Precepts. [Donna Gabai (lead) · Aditya Bhaskara · Will Clarkson · Rob Dockins · Michael Golightly · Thomas Mason · Chris Park · JP Singh]
- 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 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.
Course Materials

Course website. [www.princeton.edu/~cos126]
- Submit assignments, check grades.
- Programming assignments.
- Lecture slides. Skim before lecture; read thoroughly afterwards.
- Required readings. Sedgewick and Wayne. Intro to Programming in Java: An Interdisciplinary Approach. [Labyrinth Books]
- 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.

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

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
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.

SENDMONEY

encrypt
decrypt

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.
One-Time Pad Encryption

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

Conversion to N bits:
- SENDMONEY
  - Message: 010010 000100 001101 000011 001110 000011 001100 011000

Random Bits:
- 110010 010011 110110 110001 010101 111011 100010 111111

XOR:
- 010010 000100 001101 000011 001110 000011 001100 011000
  - XOR: 111000 100010 111111 010010

Encrypted Message:
- gX76W3v7K
**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.
- Use same $N$ random bits (one-time pad).

<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>
<tr>
<td></td>
<td></td>
<td></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>010111</td>
<td>111111</td>
<td>111011</td>
<td>110100</td>
<td>001010</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>X</td>
<td>23</td>
<td>010111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></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>010111</td>
<td>111111</td>
<td>111011</td>
<td>110100</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>110010</td>
<td>010011</td>
<td>110110</td>
<td>111011</td>
<td>010110</td>
<td>111111</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>110010</td>
<td>010011</td>
<td>110110</td>
<td>111011</td>
<td>010110</td>
<td>111111</td>
<td>110111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>
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.

Why Does It Work?

Crucial property. Decrypted message = original message.

Why is crucial property true?
- Use properties of XOR.
  - $(a \oplus b) \oplus b = a \oplus (b \oplus b) = a \oplus 0 = a$
  - associativity of $\oplus$
  - always 0
  - identity

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>

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>Y</td>
<td>24</td>
<td>011000</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.

Why Does It Work?

Crucial property. Decrypted message = original message.

Why is crucial property true?
- Use properties of XOR.
  - $(a \oplus b) \oplus b = a \oplus (b \oplus b) = a \oplus 0 = a$
  - associativity of $\oplus$
  - always 0
  - identity

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>
**Goods and Bads of One-Time Pads**

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

  eavesdropper Eve sees only random bits

**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.

instead of identical large one-time pads

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.

**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)
**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 Challenge 1**

**Goal.** Decrypt/encrypt 1,000 characters. Can we use an 11-bit LFSR?

**A.** Yes, no problem.

**B.** No, the bits it produces are not truly random.
- True, but that’s beside the point.

**C.** No, need a longer LFSR.
- Only $2^{11}$ bit patterns for register.
- "Random" bits cycle after $2^{11} - 1 = 2047$ steps.

**Lesson.** LFSR are scalable: 20 cells for 1 million bits; 30 cells for 1 billion. (but need theory of finite groups to know where to put taps)

**LFSR Challenge 2**

**Goal.** Decrypt/encrypt 1 gigabyte movie. How big an LFSR?

**A.** 30 bits should be enough.
- Probably not; Eve can try all $2^{30}$ possibilities and see which one results in a movie.

**B.** 100 bits is safe.
- Maybe; Eve would need $10^{12}$ centuries to try all $2^{100}$ possibilities.

**C.** 1000 bits makes it sufficiently secure.
- Experts have cracked LFSR.
- More complicated machines needed.

**Random Numbers**

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

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

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

```
#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);
                  j=m(4)17 ^ m(3)9 ^ k* 2-k%8 ^8,a =0,c=26;
                  for(s[y] -=16; --c; j *=2) a=a*2^i&1, i=i/2^j&1
                  <<24; for(j=127; ++y;))
        y = y+i/8<<4+12,
        s[i]=y+1<<14.^s[i]=y+i/8<<16;
        for(j=0; j<256; j++) s[j]=y+1<<16;
    ++ycm.cmc;
    if(s[y])
    {y = y+i/8<<4+12,
     s[i]=y+1<<14.^s[i]=y+i/8<<16;
     for(j=0; j<256; j++) s[j]=y+1<<16;
    }
```

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

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.

```
public class LFSR {
  private int fill[];
  private int tap;
  private int N;
  public LFSR(String fill, 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
  00110101001110101101011101010001
  00110101001110101101000001
  00110101001110101100000000
...```

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 Question

Q. What is a random number?
LFSR does not produce random numbers.
- It is a very simple deterministic machine.
- But it is hard to distinguish the bits it produces from random ones.

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