Overview

What is COS 126?
- Broad introductory survey course.
  - no prerequisites (although previous programming helpful in beginning)
- Basic principles of computer science.
  - hardware and software systems
  - programming in C and other languages
  - algorithms and data structures
  - theory of computation
  - applications to solving scientific problems

What isn’t COS 126?
- A programming course.

The Usual Suspects

Lectures: (Adam Finkelstein, Kevin Wayne)
- Tuesday, Thursday 9:00-9:50, McCosh 46.

Precepts: (Andrew Appel, Adam Finkelstein, Jason Perry, Kevin Wayne, Lisa Worthington)
- Friday - tips on assignments, clarify lecture material.
- Monday - review exercises, clarify lecture material.

All pre-registered students are assigned to a precept at the time requested.
- Note: there are 2 precepts at 11am, and 3 at 1:30pm.
- Check the Web page to see which one you are in and where it meets.
- We will only allow switching to under-subscribed precepts.
- See Kevin (Room 207) if you are not in a precept.

Required Readings

Course packet.
- Pequod copy.

Kernighan and Ritchie.
- Bible for C programmers.

Deitel and Deitel.
- “Required” C text for beginners.

Sedgewick.
- Algorithms and data structures.
- Also required in COS 226.
Lecture Outline

Programming Fundamentals (6 lectures).

Machine architecture (5 lectures).

Advanced programming (2 lectures).

Theory of computation (6 lectures).

Java (2 lectures).

Perspective (1 lecture).

R1. Course summary.

Programming Assignments

Programming assignments (designed to illustrate scientific applications):

0. Hello world

Getting started in C and Unix.
Due Thursday, February 3 at 11:59pm.
Sign up for CS101 lab reservation in class today.
(this assignment only).

Grading

Assignments (33%).
- 9 programming assignments.
- Exercises (solutions provided).

Midterms (33%).
- 2 midterms (33% total).
- Many questions drawn from exercises.

Final (34%).

Survival Guide

Keep up with the course material.
- Attend lectures and precepts.
- Do readings when assigned.
- Do exercises and understand solutions.
- Start on programming assignments early.
- Think before you write code; compose first, then write code.

Visit course home page regularly for announcements and supplemental information:

courseinfo.Princeton.EDU/courses/COS126_S2000
www.Princeton.EDU/~cs126

- Contact Kevin (Room 207) if you aren’t on course list.
- Contact CIT if you don’t remember your Unix password.
Survival Guide

Keep in touch.
- Email: your preceptor, lecturers.
- Office hours: your preceptors, other preceptors, instructors.
- Discussion group on course web page.

Ask for help when you need it!
- Preceptors, instructors, lab TA's.

What Is Computer Science?

What is computer science?
1. The science of manipulation “information.”
2. Designing and building systems that do (1).

Why we learn CS.
- Appreciate underlying principles.
- Understand fundamental limitations.

An example: linear feedback shift register machine.
- How to make a simple machine.
- What we can do with it.
- Science behind it.

Encryption Machine

Goal: design a machine to encrypt and decrypt data.

Goal

\[ \text{SENDMONEY} \]
\[ \text{W?MREAFBZ} \]
\[ \text{SENDMONEY} \]

encrypt

decrypt

Simple Encryption Scheme

1. Convert text input to N digit binary number.
2. Generate N random bits.
3. Take bitwise XOR of two strings.
4. Convert binary back into text.

Conversion

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>00001</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>00010</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>00011</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>26</td>
<td>11010</td>
</tr>
</tbody>
</table>

\[ \text{SENDMONEY} \]
\[ \begin{array}{cccccccc}
10010 & 00101 & 01100 & 00100 & 01101 & 01100 & 01101 & 11001 \\
00100 & 11001 & 00001 & 10101 & 01000 & 01111 & 01010 & 00111 & 00101 \\
10110 & 11100 & 01101 & 10001 & 00101 & 00001 & 00110 & 00010 & 11100 \\
\end{array} \]
\[ \begin{array}{cccccccc}
\end{array} \]
Decryption Scheme

1. Convert encrypted message to binary.
2. Use same N random bits.
3. Take bitwise XOR of two strings.
4. Convert binary back into text.

<table>
<thead>
<tr>
<th>W</th>
<th>?</th>
<th>M</th>
<th>R</th>
<th>E</th>
<th>A</th>
<th>F</th>
<th>B</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10110</td>
<td>11100</td>
<td>01101</td>
<td>10001</td>
<td>00101</td>
<td>00001</td>
<td>00110</td>
<td>00010</td>
<td>11100</td>
</tr>
<tr>
<td>00100</td>
<td>11001</td>
<td>00001</td>
<td>10101</td>
<td>01000</td>
<td>01111</td>
<td>01010</td>
<td>00111</td>
<td>00101</td>
</tr>
<tr>
<td>10010</td>
<td>00101</td>
<td>01100</td>
<td>00100</td>
<td>01110</td>
<td>01100</td>
<td>00101</td>
<td>11001</td>
<td></td>
</tr>
</tbody>
</table>

Why Does It Work?

Notation:
- \(a\) original message
- \(b\) random bits
- ^ XOR operation
- \(a \^ b\) encrypted message
- \((a \^ b) \^ b\) decrypted message

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

Why is crucial property true?
- \(b \^ b = 0\)
- \(a \^ 0 = a\)
- \((x \^ y) \^ z = x \^ (y \^ z)\)
- \((a \^ b) \^ b = a \^ (b \^ b) = a \^ 0 = a\)

Random Numbers

Are these 2000 numbers random?

If not, what is the pattern?

Linear Feedback Shift Register

How might the “random number machine” be built?
- We’ll investigate a simple machine called a “linear feedback shift register”.

Some terminology
- Bit: a student who is either male or female (0 or 1).
- Cell (storage element): a seat that holds one student.
- Register: whole row of students.
- Shift register: when clock strikes, stand up and take seat to left.
Linear Feedback Shift Register

Linear feedback shift register.
- Machine consists of 11 bits.
- Bit values change at discrete time points.
- Bit values at time $T+1$ completely determined by bit values at time $T$.
  - new bits 1 - 10 are old bits 0 - 9
  - new bit 0 is XOR of previous bits 3 and 10
  - output bit 0

\[
\begin{array}{cccccccccc}
    \text{a_10} & \text{a_9} & \text{a_8} & \text{a_7} & \text{a_6} & \text{a_5} & \text{a_4} & \text{a_3} & \text{a_2} & \text{a_1} & \text{a_0} \\
\end{array}
\]

Time $T$

\[
\begin{array}{cccccccccc}
    \text{a_3} & \text{a_9} & \text{a_7} & \text{a_6} & \text{a_5} & \text{a_4} & \text{a_3} & \text{a_2} & \text{a_1} & \text{a_0} & \text{a_3 a_10} \\
\end{array}
\]

Time $T+1$

The Science Behind It

Are the bits really random?

How did the computer scientist die in the shower?

How long will it take the bit pattern to repeat itself?

Will the machine work equally well if we XOR bits 4 and 10?

How many cells do I need to guarantee a certain level of security?

Properties of Shift Register “Machine”

Clocked.
- Control: start, stop, load.
- Data: initial values of bits (fill).
- Built from simple components.
- “Clock” (regular electrical pulse).
- Shift register cell remembers value until clock “ticks.”
- Some wires “input”, some “output.”

Scales to handle huge problems.
- 10 cells yields 1 thousand “random” bits.
- 20 cells yields 1 million “random” bits.
- 30 cells yields 1 billion “random” bits.
- BUT, need to understand abstract machine!
  - higher math needed to know XOR taps

Properties of Computers

Same basic principles as LFBSR:
- Clocked.
- Control: start, stop, load.
- Data: initial values of bits.
- Built from simple components.
- Scales to handle huge problems.

Abstraction aids in understanding.
Simulating The Abstract Machine

C program to produce “random” bits.

```c
#include <stdio.h>
#define N 100

int main(void) {
    int i, new, fill = 01502;
    for (i = 0; i < N; i++) {
        new = ((fill >> 10) & 1) ^ ((fill >> 3) & 1);
        fill = (fill << 1) + new;
        printf("%d\n", new);
    }
    return 0;
}
```

You’ll understand this program by next week!

\[
\begin{align*}
\text{>>} & \quad \text{shift right} & \quad \& \quad \text{“and” (1 if both bits 1, 0 otherwise)} \\
\text{<<} & \quad \text{shift left} & \quad ^\text{“exclusive or” (1 if both bits are different)}
\end{align*}
\]

Simulating The Abstract Machine

C program to produce “random” bits.

Any “general purpose” machine can be used to simulate any abstract machine. Implications are:

- Test out new programs.
- Use old programs.
- Understand fundamental limitations of computers.

Layers of Abstraction

Layers of abstraction (recurring theme).
- Precisely defined for simple machine.
- Use it to build more complex one.
- Develop complex systems by building increasingly more complicated machines.
- Improve systems by substituting new (better) implementations of abstract machines at any level.

Layered abstraction (LFBSR).
- Simple piece of hardware.
- Generate “random” bits.
- Use “random” bits for encryption.
- Use encryption for Internet commerce.

“Computer” layers of abstraction.
- Complex piece of hardware.
  - CPU, keyboard, printer, storage devices
- Machine language programming.
  - 0’s and 1’s
- Software systems.
  - editor (emacs): create, modify files
  - compiler (lcc): transform program to machine instruction
  - operating system (Unix): invoke programs
- Windowing system (X).
  - illusion of multiple computer systems