Overview

What is COS 126?
- Broad, but technical, intro to fundamental ideas of CS.
  - no prerequisites
  - experienced programmers, see COS 217 or COS 226

- Basic CS principles.
  - hardware, software systems
  - programming in C, other languages
  - algorithms and data structures
  - theory of computation
  - applications to solving scientific problems
  - critical thinking

What isn't COS 126?
- Solely a programming course.

The Usual Suspects

Lectures: David August (David)
- Tuesdays and Thursdays, 10:00 - 10:50, Frist 302.

Precepts: Donna Gabai (Donna)
  Kai Li (Kai)
  Lisa Worthington (Lisa)
- Fridays - tips on assignments, clarify lecture material.
- Mondays - review exercises, clarify lecture material.

Staffed lab: Undergrad lab assistants.
- Friend 016, 017.
- Schedule to be posted on Web.

Grading

Programming assignments: 34%
- Can drop lowest one.

Midterms: 33%
- 2 midterms (equal weight).
- Many questions drawn from exercises.

Final: 33%

Staff discretion.
- Adjust borderline cases.

Course grades.
- No preset curve.
- Typical breakdown

![Grading Chart]
Required Readings

- Syllabus.
- Programming assignments.
- Lecture notes.
- Old exams.
- Exercises and solutions.

King.
- Introduction to C.

Sedgewick.
- Algorithms and data structures.

Harel.
- What computers can't do.

Lecture Outline

Programming fundamentals.
- P1. C language basics
- P2. OS basics
- P3. Arrays
- P5. Structs and data types.
- P6. ADT, stacks, and queues.
- P7. Recursion I.
- P8. Recursion II.

Machine architecture.
- A1. TOY machine.
- A2. TOY programming / simulator.
- A4. Sequential circuits.
- A5. Building a TOY machine.

Advanced programming.
- P9. Linked lists and pointers.
- P10. Card game example.
- P11. Trees and database search.
Lecture Outline

Programming fundamentals.

Machine architecture.

Advanced programming.

Theory of computation.
  T1. Turing machine.
  T2. Computability.
  T3. Analysis of algorithms.
  T4. NP-completeness.

Lecture Outline

Programming fundamentals.

Machine architecture.

Advanced programming.

Theory of computation.

Theory of computation.
  T1. Turing machine.
  T2. Computability.
  T3. Analysis of algorithms.
  T4. NP-completeness.

Special topics.
  S1. Cryptology.
  S2. Artificial intelligence.

Programming Assignments

Weekly programming assignments.
  • Due Wednesday 11:59pm via electronic submission.

Operating systems.
  • Linux, OS X, Unix, Windows, . . .
  • Your code should work properly on ALL systems.

Computing equipment.
  • Your machine.
  • OIT machines.
    – Friend 016 and 017 labs
  • Assignment 0: setup C programming environment.

Survival Guide

Keep up with the course material.
  • Attend lectures and precepts.
  • Do readings when assigned.
  • Do exercises and understand solutions.
  • Plan multiple lab sessions for programming assignments.
  • Visit course home page regularly.

Ask for help when you need it!
  • Preceptors / instructors.
    – email, office hours, precepts
    – concepts, programming assignments, exercises
  • Lab TAs.
    – OS support, help with minor debugging
What's Ahead?

ASAP. Read Chapters 1-3 of King. Skim Chapters 4-6.

Thursday. Lecture: Introduction to C.

Thursday 11:59PM. Assignment 0 due.

Friday. First precept meets. Exercises: “Hello World”
   • Check course web page to see which precept you're in.
   • If not in precept, see Donna after class or this
     afternoon 1:00 - 3:00 PM in CS 205, 35 Olden Street.

Monday. Second precept meets. Exercises: C Expressions, Loops

END OF ADMINISTRATIVE STUFF

What Is Computer Science?

What is computer science?
   • The study of computation.

What is computation?
   • The process of manipulating and transforming information.

What CS is not.
   • CS is not solely programming.
   • We use programming to express CS ideas.
   • ”Computer science is no more about computers than astronomy is
     about telescopes.” — E. Dijkstra

Why we learn CS.
   • Appreciate most fundamental underlying principles.
   • Understand inherent limitations of computing.
   • What can be automated?

An example: "linear feedback shift register machine."
   • How to make a simple machine.
     – that produces pseudo-random bits
   • What we can do with it.
     – use to encrypt and decrypt secret messages
     – DeCSS program to copy DVD's!
   • Science behind it.

Encryption Machine

Goal: design a machine to encrypt and decrypt data.

\[
\begin{align*}
\text{SEND} & \quad \text{MONEY} \\
W & \quad M \quad R \quad E \quad A \quad F \quad B \\
\text{SEND} & \quad \text{MONEY}
\end{align*}
\]

encrypt

decrypt
Encryption Machine

Goal: design a machine to encrypt and decrypt data.

SENDMONEY

encrypt

W ? M R E A F B ?

decrypt

SENDMONEY

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.

Simple Encryption Scheme (One-Time Pad)

1. Convert text input to N bits.
2. Generate N random bits (secret key).

SENDMONEY

10010 00101 01100 00100 01101 01110 01100 00101 11001

00100 11001 00001 10101 01000 01111 01010 00111 00101

A Russian One-Time Pad
Simple Encryption Scheme (One-Time Pad)

1. Convert text input to N bits.
2. Generate N random bits (secret key).
3. Take bitwise XOR of two strings.
   - Sum pair of bits (1 if sum is odd, 0 if even)

<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>message</th>
<th>binary</th>
<th>random bits</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEND</td>
<td>10010</td>
<td>001001</td>
<td>101</td>
</tr>
<tr>
<td>MONEY</td>
<td>01100</td>
<td>00001</td>
<td>101</td>
</tr>
<tr>
<td>Y</td>
<td>11001</td>
<td>001001</td>
<td>011</td>
</tr>
<tr>
<td>Z</td>
<td>11010</td>
<td>000101</td>
<td>001</td>
</tr>
</tbody>
</table>

Decryption Scheme (One-Time Pad)

1. Convert encrypted message to binary.

<table>
<thead>
<tr>
<th>encrypted</th>
<th>binary</th>
<th>converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>W ? M R E A F B ?</td>
<td>10110 11100 011010001010001101000110100011011100</td>
<td>A 100001</td>
</tr>
</tbody>
</table>

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>Y</td>
<td>25</td>
<td>11001</td>
</tr>
<tr>
<td>Z</td>
<td>26</td>
<td>11010</td>
</tr>
</tbody>
</table>
Decryption Scheme (One-Time Pad)

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

<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>Y</td>
<td>25</td>
<td>11001</td>
</tr>
<tr>
<td>Z</td>
<td>26</td>
<td>11010</td>
</tr>
</tbody>
</table>

Why Does It Work?

Notation:
- a original message
- b random bits (secret key)
- ^ 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?
- \((a ^ b) ^ b = a \land (b ^ b) = a \land 0 = a\)
Are these 2000 numbers random?

If not, what is the pattern?

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

Jon von Neumann (left), ENIAC (right)

How might the "random number machine" be built?

- "Linear feedback shift register."
- "Linear congruential generator."
  - see Assignment 1

Some terminology

- Bit: 0 or 1.
- Cell: storage element that holds 1 bit.
- Register: array of cells.
- Shift register: when clock ticks, bits propagate one position to left.

Linear feedback shift register.

- Machine consists of 11 bits.
- Bit values change at discrete time points.
- Bit values at time T+1 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

LFBSR Demo
Random Numbers

Are these 2000 numbers random?

```
0101100100000110001001011010110111001001111011010010001011111010100111100111001101001011010
110011010111010110001011100111001000101001000100000101101101110110101101011011011010101101
111010000011101101011001011011011010111011100110110011011101111011000010110110111011011001
1101010100101100001001101011011101101011011111010101110101001110111011111111101111011000
11011000110010001111110110110111010100111111001100110011101101110111011011111110010011111
1011011011011010000101101110110001101100011011001101111100001111110000111000110110111111
11110101011101101110110111011101100110110111011101110111111111011110111111000010000000000
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
111101010111011011111011101110111011111111111101110110111111111011110111111001100011011101
```

No. This is output of LFBSR!

Are the bits really random?

No! Real machines are deterministic.

How did the computer scientist die in the shower?

The instructions on the shampoo read “lather, rinse, repeat.”

Will bit pattern repeat itself?

Yes, after $2^{11} - 1 = 2047$ steps.

What if I need more bits?

Scalable: 20 cells for 1 million bits, 30 for 1 billion.

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

No, need to understand theory of abstract rings.

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

Subject of active research.

The Science Behind It

"General Purpose Computer"

Same basic components.

- **Control:** start, stop, load.
- **Clock:** regular pulse that triggers events.
- **Memory:** shift register cell remembers value until clock “ticks.”
- **Input:** initial values of bits.
- **Computation:** shifting and XORing the bits.
- **Output:** sequence of pseudo-random bits.

Same important properties.

- Built from simple components.
- Scales to handle huge problems.

Need deeper understanding of “computer” to use effectively.

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

"Linear Feedback Shift Register "Machine"

Basic components.

- **Control:** start, stop, load.
- **Clock:** regular pulse that triggers events.
- **Memory:** shift register cell remembers value until clock “ticks.”
- **Input:** initial values of bits (seed).
- **Computation:** shifting and XORing the bits.
- **Output:** sequence of pseudo-random bits.

Important properties.

- Built from simple components.
- 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 deep understanding of abstract machine!

"General Purpose Computer"

Same basic components.

- **Control:** start, stop, load.
- **Clock:** regular pulse that triggers events.
- **Memory:** remembers value until clock “ticks.”
- **Input:** initial values of bits.
- **Computation:** transform input in various ways.
- **Output:** result of computation.

Same important properties.

- Built from simple components.
- Scales to handle huge problems.

Need deeper understanding of “computer” to use effectively.

Critical difference. General purpose machine can be programmed to simulate ANY abstract machine.
Simulating The Abstract Machine in C

Produces exactly same bits as LFBSR.

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

int main(void) {
    int i, new;
    int b10 = 0, b9 = 1, b8 = 1, b7 = 0, b6 = 1, b5 = 0;
    int b4 = 0, b3 = 0, b2 = 0, b1 = 1, b0 = 0;

    for (i=0; i < N; i++) {
        new = b3 ^ b10;
        b10 = b9; b9 = b8; b8 = b7; b7 = b6; b6 = b5;
        b5 = b4; b4 = b3; b3 = b2; b2 = b1; b1 = b0; b0 = new;
        printf("%d", new);
    }
    return 0;
}
```

You'll understand this program by next week.

A Machine That Generates "Random" Numbers

C program to produce "random" bits using bit operations.

```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;
}
```

Simulating The Abstract Machine

Lecture I1: Supplemental Notes

01001100100000011000100010111010101111001001111001110100100001101111111100101101
0011011011001010011111101010010110001100
011101101110000011010001000010101000 ...
Layers of Abstraction: LFBSR

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.

LFBSR layers of abstraction.
- Simple piece of hardware.
- Generate "random" bits.
- Use "random" bits for encryption.
- Use encryption for Internet commerce.
Layers of Abstraction: Computer

"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 (gcc): transform program to machine instruction
  - operating system (Unix): invoke programs

- Windowing system (X).
  - illusion of multiple computer systems