Lecture I1: Introduction

COS 126
Princeton University
Fall 2001

Bob Sedgewick
Kevin Wayne

Overview

What is COS 126?
- Broad, but technical, intro to fundamental ideas of CS.
  - no prerequisites
    (although previous programming very helpful in beginning)
- 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: (Bob Sedgewick, Kevin Wayne)
- Tuesday, Thursday 10:00 - 10:50, Frist 302.
- Lectures extend into reading period.

Precepts: (Donna Gabai, Kevin Wayne)
- Friday - tips on assignments, clarify lecture material.
- Monday - review exercises, clarify lecture material.
- Introductory precept meets Friday.
- If not in precept, see Kevin after class or this afternoon at 4:30 - 5 in CS 207, 35 Olden Street.

Lab Assistants: (many fine Princeton undergrads)
- Lab TA schedule to be posted on Web.

Grading

Assignments: 34%
- Weekly programming assignments.
- Exercises (solutions provided).

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

Final: 33%
- Staff discretion.

Course grades.
- No preset curve.
- Last year’s breakdown.
Required Readings

Course packet.
- Pequod copy (U-Store, 36 University Place).
- Syllabus.
- Programming assignments.
- Lecture notes.
- Old exams.
- Exercises.
- Solutions to exercises.

King.
- Intro to C.

Sedgewick.
- Algorithms and data structures.

Lecture Outline

Programming fundamentals (7 lectures).

Machine architecture (5 lectures).

Advanced programming (3 lectures).

Theory of computation (5 lectures).

Systems (3 lectures).

Perspective (1 lecture).

Programming Assignments

Weekly programming assignments.
- Usually due Wed 11:59pm via electronic submission.

Assignment 0 due THIS FRIDAY.
- Setup a C programming environment.
  - Linux, OS X, Unix, Windows
- Note: all code should work properly on all systems.

CIT computing clusters in Friends Center.
- Bring your laptop and plug-in to Internet; or use "arizona" machines running Unix.
- Staffed by CS lab TAs.
- Go to 87 Prospect if you don’t know login / password.

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 for announcements and supplemental information: www.Princeton.EDU/~cs126

Ask for help when you need it!
- Preceptors / instructors: (email, office hours, precepts)
  - concepts, programming assignments, exercises
- Lab TAs: (on duty in Friends Center lab)
  - OS support, help with minor debugging

END OF ADMINISTRATIVE STUFF
What Is Computer Science?

What is computer science?
- The study of computation.

What CS is not.
- CS is not solely programming.
- Programming is a useful tool to express CS ideas.

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

What Is Computer Science?

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.

```
SENDMONEY
W MREAFBZ
```

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.

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)

```
XOR Truth Table
x y x ^ y
0 0 0
0 1 1
1 0 1
1 1 0
```

```
message
binary
random bits
XOR
```

```
SENDMONEY
```

```
10110 00101 01100 00100 01101 01110 01100 00101 11001
00100 11001 00001 10101 01000 01111 01010 00111 00101
01110 11100 01101 10001 10010 00001 00110 00010 11100
```
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.
4. Convert binary 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>...</td>
<td>...</td>
<td>...</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.

Why Does It Work?

Notation:

- \(a\) original message
- \(b\) random bits (secret key)
- \(^\wedge\) XOR operation
- \((a \wedge b)\) encrypted message
- \((a \wedge b) \wedge b\) decrypted message

Crucial property: \((a \wedge b) \wedge b = a\).

- Decrypted message = original message.

Why is crucial property true?

- \((a \wedge b) \wedge b = a \wedge (b \wedge b) = a \wedge 0 = a\)

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?
- "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

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

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.

The Science Behind It

Are the bits really random?

How did the computer scientist die in the shower?

Will bit pattern 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?

Linear Feedback Shift Register "Machine"

Basic components.
- **Control**: start, stop, load.
- **Clock**: regular electrical pulse that triggers events.
- **Memory**: shift register cell remembers value until clock "ticks."
- **Input**: initial values of bits (seed).
- **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 electrical pulse that triggers events.
- **Memory**: remember values until clock "ticks."
- **Input**: initial values of bits.
- **Output**: result of computation.

Same important properties.
- Built from simple components.
- Scales to handle huge problems.

Need deep 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;
}
```

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