Lecture l1: Introduction

COS 126
Princeton University
Fall 2000

Bob Sedgewick
Kevin Wayne

Overview

What is COS 126?

- Broad introductory survey course.
  - 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?

- A programming course.

The Usual Suspects

Lectures: (Bob Sedgewick, Kevin Wayne)

- Tuesday, Thursday 10:00 - 10:50, Frist 302.
- Extends into reading period.

Precepts: (Ben Gum, Christine Lv, Kevin Wayne, Lisa Worthington)

- Friday - tips on assignments, clarify lecture material.
- Monday - review exercises, clarify lecture material.

Undergraduate Coordinator: (Tina McCoy)

- CS Building, Room 410.

Computer Lab Assistants: (many fine Princeton undergrads)

- Public Unix lab in CS 101.
- Lab TA schedule to be posted on Web.

Signing Up for a Precept

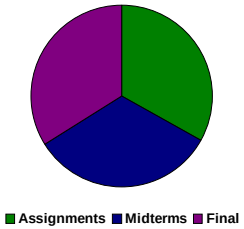
Everyone must be enrolled in one precept.

- All pre-registered students already set - check Web page.
- If not in precept, see Kevin after class or this afternoon at 4 - 4:30 in Room 207.
- Note: multiple precepts at certain times.
- Introductory precept meets Friday.

Grading

Assignments: 33%

- 9 programming assignments.
- Exercises (solutions provided).



Midterms: 33%

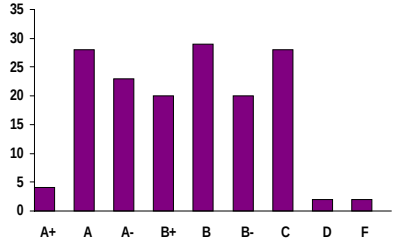
- 2 midterms (33% total).
- Many questions drawn from exercises.

Final: 34%

Staff discretion.

Course grades.

- No preset curve.
- Last year's breakdown.



Where To Program

Public cluster in Room 101, CS Building.

- 30 Sun Ultra 5 machines running Unix.
- Lab run by CIT.
  - go to 87 Prospect Ave if you don't have an account or don't know password
- Supported by CS lab assistants.

Can I work from home?

- Use home PC as terminal:
  - telnet to arizona
  - need X-Windows emulator for GUI
- Use home PC as primary computer:
  - Linux
  - Windows / Mac OS
- All code must work properly on arizona.

Required Readings

Course packet.

- Pequod copy (6 Nassau Street).
- 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 (6 lectures).

Systems (3 lectures).

Perspective (1 lecture).

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.

Visit course home page regularly for announcements and supplemental information:

`courseinfo.Princeton.EDU/courses/COS126_F2000`
`www.Princeton.EDU/~cs126`

Survival Guide

Keep in touch.

- Email: your preceptor, instructor.
- Office hours: your preceptor, other preceptors, instructors.
- Discussion group on course web page.

Ask for help when you need it!

- Preceptors, instructors: concepts, programming assignments, exercises.
- Lab TA's: Unix support, help with debugging.

END OF ADMINISTRATIVE STUFF

What Is Computer Science?

What is computer science?

1. The science of manipulating "information."
2. Designing and building systems that do (1).

What CS is not.

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

Why we learn CS.

- Appreciate most fundamental underlying principles.
- Understand inherent limitations.
- 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

Bit = 0 or 1

- What we can do with it.
  - use to encrypt and decrypt secret messages
- Science behind it.

Encryption Machine

Goal: design a machine to encrypt and decrypt data.

S	E	N	D	M	O	N	E	Y

encrypt

W	?	M	R	E	A	F	B	Z

decrypt

S	E	N	D	M	O	N	E	Y

Simple Encryption Scheme (One-Time Pad)

1. Convert text input to N bits.

Conversion

char	dec	binary
A	1	00001
B	2	00010
…	…	…
Y	25	11001
Z	26	11010

S	E	N	D	M	O	N	E	Y

message

10010	00101	01100	00100	01101	01110	01100	00101	11001

binary

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

S	E	N	D	M	O	N	E	Y

message

10010	00101	01100	00100	01101	01110	01100	00101	11001

binary

00100	11001	00001	10101	01000	01111	01010	00111	00101

random bits

10110	11100	01101	10001	00101	00001	00110	00010	11100

XOR

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.

Conversion

char	dec	binary
A	1	00001
B	2	00010
…	…	…
Y	25	11001
Z	26	11010

S	E	N	D	M	O	N	E	Y

message

10010	00101	01100	00100	01101	01110	01100	00101	11001

binary

00100	11001	00001	10101	01000	01111	01010	00111	00101

random bits

10110	11100	01101	10001	00101	00001	00110	00010	11100

XOR

W	?	M	R	E	A	F	B	?

send

Decryption Scheme (One-Time Pad)

1. Convert encrypted message to binary.

Conversion

char	dec	binary
A	1	00001
B	2	00010
...	...	...
Y	25	11001
Z	26	11010

W	?	M	R	E	A	F	B	?	message
10110	11100	01101	10001	00101	00001	00110	00010	11100	binary

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.

Conversion

char	dec	binary
A	1	00001
B	2	00010
...	...	...
Y	25	11001
Z	26	11010

W	?	M	R	E	A	F	B	?	message
10110	11100	01101	10001	00101	00001	00110	00010	11100	binary
00100	11001	00001	10101	01000	01111	01010	00111	00101	random bits
10010	00101	01100	00100	01101	01110	01100	00101	11001	XOR
S	E	N	D	M	O	N	E	Y	send

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?

- 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?

```
010011001000000011000100010111010101111001001111001110100100001101111111110010110100110110
110010100111111010100101100011000111011011100000110100010000101010001001110011011110011 1
110000001111011101101000011010111011011000010111101110010110000100111111101101001011110
101000011100100010110010001101010001011111011111000010110101010000101000000001010101010 1
111101011110000001011011101100000111000100011111101111000011010011001010001110101110100 1
110000010010101010001101100010010010001100010101001100011011110111101001001011111101100 0
011111001100001110100010010100010101111111010110100100111110110010110110111110010100101 1
100100000111010101011011000110101100111110100011110010001111011001111001010110101000001 1
011001100000011000000000110011001101010011010111111001001011011111101000011110101010010 10
000010011101110011110100110100111010000000111001100111101101111000111000010000110011101 11
101101011010110001110010101001000000101011101111101001111000110000110010101011001001110 00
101011011100100100101011100111001011100000010110011001001000010011011101010011110101110 0
001010010001010001100100110000000100100010000000100000000001000100010011000100110101011 100
011011010101100000101001100111001111110001011010001010011101100010110000000101110111010 1
101101011110100000111110111010010100100110110010000101110011000111111001101001010110001 0
100001000000100011001101110001011110011011010001110001100101110100011010000000110111011 10
001111100010010110011101011001011111001110000110000100011101111111100011110000000011111 111
111000011110001000011101100110100001001001100100000011000100010111010101111001001111001 1
101001000011011111111001011010011011011001010011111101010010110001100011101101110000011 0
100010000101010001001110011011110011111000000111101110110100001101011101101100001011110 1
110010110000100111111110110100101111010100001110010001011001000110101000101111101111100 0
010110101010000101000000001010101010111110101111000000101101110110000001110001000111111 101
11100001101001100101000111010111010011100000100101010100011101100...
```

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

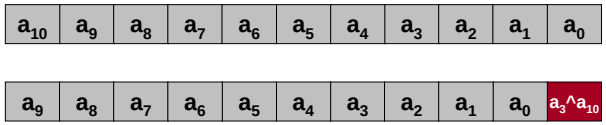
Some terminology

- 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



LFBSR Demo

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?

Properties of Shift Register "Machine"

Clocked.
Control: start, stop, load.
Data: initial values of bits (seed).
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 in C

Produces exactly same bits as LFBSR.

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; b3 = b2; b2 = b1; b1 = b0; b0 = new;
    printf("%d", new);
  }
  return 0;
}
```

You'll understand this program by next week.

^ means XOR in C

```
01001100100000011000100010111010101111001
10011110011101001000011011111111100101101
00110110110010100111111010100101100011000
01110110111000001101000100001010100 ...
```


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.

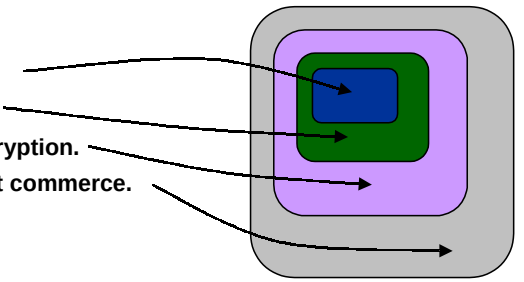
Layers of Abstraction: LFBSR

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

Lecture I1: Supplemental Notes

Simulating The Abstract Machine

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

lfbsr.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;
}
```

octal constant

- >> shift right
- << shift left
- & "and" (1 if both bits 1, 0 otherwise)
- ^ "exclusive or" (1 if bits are different)