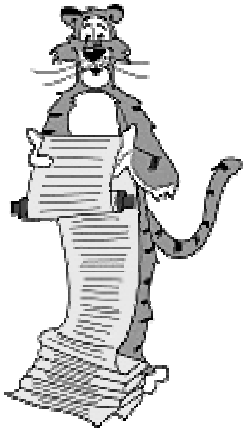
Lecture I1: Introduction



COS 126
Princeton University
Spring 2001

Randy Wang

Overview

What is COS 126?

- Broad, but technical, intro 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: (Randy Wang)

- Tuesday, Thursday 10:00 - 10:50, Frist 302.

Precepts: (Doug Clark, Matt Webb, 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

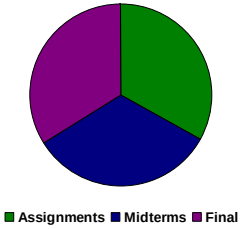
Everyone must be enrolled in one precept.

- All pre-registered students already set - see list outside.
- If not in precept, see Kevin after class
  or this afternoon at 4:30 - 5 in CS 207, 35 Olden Street.
- Introductory precept meets Friday.
- Note: multiple precepts at certain times.
  - check course Web site to see which one you are in

Grading

Assignments: 33%

- Programming assignments.
- Exercises (solutions provided).



Midterms: 33%

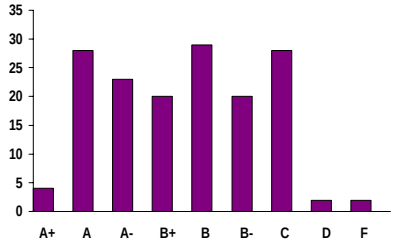
- 2 midterms (33% total).
- Many questions drawn from exercises.

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 (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).

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:

`courseinfo.Princeton.EDU/courses/COS126_S2001`
`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 of computing.
- What can be automated?

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

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.
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?

```
01001100100000001100010001011101010111100100111100111010010000110111111110010110100110110
```

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**

a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Time T
a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	a_3^a_{10}	Time T+1

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

```
0100110010000001100010001011101010111100
1001111001110100100001101111111100101101
0011011011001010011111101010010110001100
0111011011100000110100010000101010000 ...
```


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.

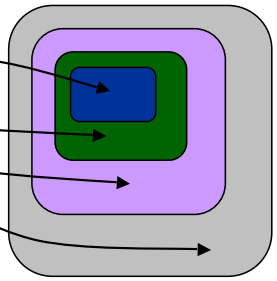
Layers of Abstraction: LFBSR

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.

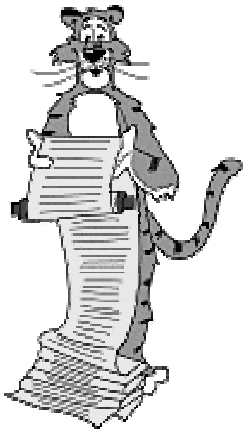


Layers of Abstraction: Computer

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