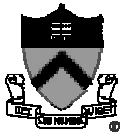
Lecture I1: Introduction



COS 126
Princeton University
Spring 2002

Doug Clark
Andrew Appel, Bob Dondero,
Donna Gabai, Lisa Worthington

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: (Doug Clark)

- Tuesdays and Thursdays, 10:00 - 10:50, Friend 101

Precepts: (Andrew Appel, Bob Dondero, Donna Gabai, Lisa Worthington)

- Fridays - tips on assignments, clarify lecture material.
- Mondays - review exercises, clarify lecture material.
- First precept meets this Friday.
- Check course Website 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.

Lab Assistants: (many fine Princeton undergrads)

- Lab TA schedule to be posted on Web.

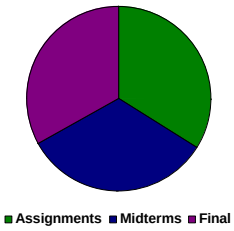
Grading

Assignments: 34%

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

Midterms: 33%

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

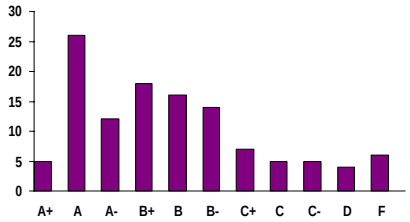


Final: 33%

Staff discretion.

Course grades.

- No preset curve.
- Last spring's breakdown →



Required Readings

Course packet (at Pequod Copy in U-store).

- 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 (5 lectures).

Machine architecture (5 lectures).

Advanced programming (5 lectures).

Theory of computation (5 lectures).

Special topics (2 lectures).

Perspective (1 lecture).

Programming Assignments

Weekly programming assignments.

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

Assignment 0 due TOMORROW.

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

OIT computing clusters in Friend 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.

For assistance with Assignment 0, go to Friend Center Lab today or tomorrow.

Survival Guide

Keep up with the course material.

- Attend lectures and precepts.
- Do readings when assigned (or earlier!).
- 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 Friend 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 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.

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

Bit = 0 or 1

- 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.

S	E	N	D	M	O	N	E	Y

encrypt

W	?	M	R	E	A	F	B	?

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.
   - 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	
10010	00101	01100	00100	01101	01110	01100	00101	11001	message
00100	11001	00001	10101	01000	01111	01010	00111	00101	binary
10110	11100	01101	10001	00101	00001	00110	00010	11100	random bits
									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	?	encrypted

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	?	encrypted
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	message

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 ^ (b ^ b) = a ^ 0 = a

Random Numbers

Are these 2000 numbers random?

```
0100110010000001100010001011101010111100100111100111010010000110111111111001011010011011
1100010011111101010010110001100011101101110000011010001000010101000100111001101111001111
1100000011110111011010000110101110110110000101111011100101100001001111111101101001011110
1010000111001000101100100011010100010111101111100001011010101000010100000001010101010 1
1111010111000001011011011000001110001000111111101111000011010011001010001011011011001
1100000100101010100011011000100100100011000101010011000110111011110100100101111110110 00
0111110011000011101000100101000101011111110101101001001111101100101101111100101001011
1001000001110101010110110001101011001111101000111100100011101100111001101010110100000011
0110011000001100000000011001100110101001101011111100100101101111110100001111010101001010
0000100111011100111101001101001110100000011100110011101101110001110000100001100111011 1
1011010110101100011100101010010000010101110111101001111000110000110010101011001000011100
1010110111100100100101011100111001011100000010110011001001000010011011101010011101011100
0010100100010100011001001100000010010001000000100000000000100010001001100010011010111 00
0110110101101100001010011001110011111100010110100010100111011000101100000001011011101 01
1011010111101000001111011101001010010011011001000010111001100011111100110100101011000 10
1000010000010001100110111000101111001101101000111000110010111010001101000000011011011 10
0011111000100101100111010110010111110011000010000100011101111111000111100000001111111 1
1110000111100010000111011001101000010010011001000000110001000101110101011110010011110011
1010010000110111111110010110100110110110010100111111010100101100011000111011011100000 110
1000100001010100010011100110111100111110000001111011101010000110101110101011000010111101
1100101100001001111111110110100101111010100001110010001011001000110101000101111011111000
0101101010100001010000000010101010111110101111000001011011101100000111000100011111110 1
1110000110100110010100011101011101001110000010010101010001101100...
```

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

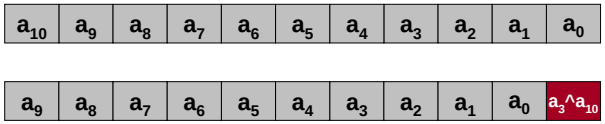
Some terminology

- 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?

fill in answer

How did the computer scientist die in the shower?

Will bit pattern repeat itself?

What if I need more bits?

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 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 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 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.

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

```
01001100100000011000100010111010101111001001111001110100100001101111111100101011001101101100101001111110101001011000110001110110111000001101000100001010100 ...
```

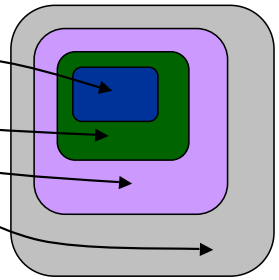

Layers of Abstraction: LFBSR

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)**