COS 126 Lecture 1: Introduction

Introductory survey course
  no prerequisites
  basic principles of computer science
  learn to use computers effectively
  check FAQs on web

Topics introduced:
  hardware and software systems
  programming in C and other languages
  algorithms and data structures
  theory of computation
  applications to solving scientific problems

#include <stdio.h>
main()
{
  printf("This is a C program\n");
}

Q. How did the computer scientist die
   in the shower?
A. The instructions on the shampoo said
   "Lather, Rinse, Repeat"
Lecture Outline

INTRODUCTION (1 lecture)
   I1. Abstract machine example

PROGRAMMING FUNDAMENTALS (7 lectures)
   P1. C
   P2. Unix
   P3. Arrays/structs/lists
   P4. Card game example
   P5. ADTs
   P6. Recursion
   P7. Trees

ARCHITECTURE (5 lectures)
   A1. TOY
   A2. TOY/simulator
   A3. Boolean logic
   A4. Sequential circuits
   A5. Machine organization
Lecture Outline (continued)

THEORY OF COMPUTATION (6 lectures)
  T1. REs and FSAs
  T2. Turing machines
  T3. Formal languages
  T4. Computability
  T5. Algorithms/Complexity
  T6. NP-completeness

SYSTEMS (5 lectures)
  S1. Compilers
  S2. Operating systems
  S3. Applications
  S4. Java
  S5. Java/Graphics

REVIEW (2 lectures)
  R1. History
  R2. Course review
COS 126 Survival Guide

Participate in precepts
  Friday: programming assignments/review
  Monday: quizzes/exercises

Keep up with the course materials
  read over handouts when you get them
  www.CS.Princeton.EDU/courses/cs126
  prepare for precepts

Keep in touch
  mail
  office hours
  after class

Use the simplest tool that gets the job done

Understand your program
  what would the machine do?
  find the first bug
  develop programs incrementally
  plan multiple lab sessions

Ask for help when you need it
An Abstract Machine

Simple abstract computational device

Linear feedback shift register (LFBSR)

Machine consists of 11 BITS, or 0-1 values
Bit values change at discrete time points
Bit values at time $T+1$ completely determined by values at time $T$

```
T   10  9  8  7  6  5  4  3  2  1  0  $10^3$
T+1 10  9  8  7  6  5  4  3  2  1  0
```

“XOR” of two bits (addition mod 2)
1 if different; 0 if same

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a(^b)</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>

Magic properties:

$b^b = 0$
$a^0 = a$
$(a^b)^b = a^{(b^b)} = a^0 = a$
LFBSR example

0 1 1 0 1 0 0 0 0 1 0
1 1 0 1 0 0 0 0 1 0 0
1 0 1 0 0 0 0 1 0 0 1
0 1 0 0 0 0 1 0 0 1 0
1 0 0 0 0 1 0 0 1 0 0
0 0 0 0 1 0 0 1 0 0 1
0 0 0 1 0 0 1 0 0 1 1
0 0 1 0 0 1 0 0 1 1 0
0 1 0 0 1 0 0 1 1 0 0
0 1 0 0 1 0 0 1 1 0 0
1 0 0 1 0 0 1 1 0 0 1
0 0 1 0 0 1 1 0 0 1 0
Using "Random" Bits for Encryption

Convert message to bitstream

SENDMONEY
100100010101100001000110101110011000010111001

Send bit-by-bit XOR with "random" bitstream

SENDMONEY
100100010101100001000110101110011000010111001
00100110010001101010100001111010100011100101
101101110001101100010010100001001100001011100

Message looks random to anyone reading it

SENDMONEY
101101110001101100010010100001001100001011100
W?MREAFBZ

Receiver has identical machine
(Secretly) provide receiver with initial fill
Receiver computes XOR with SAME "random" bitstream

SENDMONEY
101101110001101100010010100001001100001011100
00100110010001101010100001111010100011100101
100100010101100001000110101110011000010111001
SENDMONEY

Works because \((a \oplus b) \oplus b = a \oplus (b \oplus b) = a\)
Properties of shift register “machine”

Clocked

Control: start, stop, or “load”
Data: initial values of bits (fill)

Built from very simple components
“clock” (regular electrical pulse)
electrically controlled 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)

Same basic principles used for computer
clocked
all built from switches with feedback
control, data
abstraction aids understanding
Simulating an Abstract Machine

C program to produce random bits

#include <stdio.h>
main()
{
    int i, new, fill = 01502;
    for (i = 0; i < 10; i++)
    {
        new = ((fill>>10 & 1)^(fill>>3 & 1));
        fill = (fill << 1) + new;
        printf("%d ", new);
    }
}

You'll understand this program by next week!
Uses C bitwise operations
    >> shift right
    << shift left
    & "and" (1 if both 1, 0 otherwise)
    ^ "exclusive or"

Any "general-purpose" machine can be used
to simulate any abstract machine. Implications:
test out new designs
use old programs
understand fundamental limitations
Layers of abstraction 
precisely define a simple machine 
use it to build a more complex one 
develop complex systems by building 
increasingly more complicated machines 
improve systems by substituting 
ew (better) implementations 
of abstract machines at any level

LFBSR layers of abstraction 
simple piece of hardware 
converts fill to "random" bits 
can use "random" bits for encryption 
can use encryption for internet commerce

"Computer" layers of abstraction 
complex piece of hardware 
CPU, keyboard, printer, storage device 
machine language programming 
software systems 
editor (emacs): create, modify files 
compiler (cc): transform program 
to machine instructions 
operating system (Unix): invoke programs 
windowing system (X): 
illusion of multiple computer systems