```
 *  **  *      **   *   * *** * * ****
*  ****  *** *  *    ** ********  * ** *
  ** ** **  * *  ****** * *  * **   **
 *** ** ***     ** *   *    * * *   *  *
**  ** **** *****      **** *** ** *
 ** * *** ** **    * **** ***  * **    *
  ******** ** *  * **** * *    ***  *
* **  *   ** * *   * ***** *****    * **
 * * *    * *        * * * * * ***** * *
***     * ** *** **     ***   *   ******
* ****    ** *  **  * *   *** * *** *  *
**     *  * * * *   ** **   *  *  *   **
** **    ***** **     *** *   *  * *   *
 * ******* * ** *  *  ***** **  * ** **
*****  * *  * ***  *     *** * * * ** **
 ****  * * ** * *     ** **  **      **
        **  **  ** * *  ** * ******  *  *
 ** ****** *    **** * * *  * *     *  *
** ***  **** *  ** *  *** *      ***  **
  **** ** ****   ***    *    **  *** ***
* ** * ** * **   ***  * * *  *     * * *
** ***** *  ****   **    **  * * * **  *
  ***   * * ** ***  *  *  * * ***  ***
```

COS 126 Lecture 1: Introduction

R. Sedgewick
rs@cs.princeton.edu

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

C

- Basics
- Functions and Modules

MACHINE ORGANIZATION

- Toy Machine
- Machine Language
- Simulator

DATA STRUCTURES

- Data Types
- Data Structures
- Recursion
- Trees

ARCHITECTURE

- Boolean Logic
- Sequential Circuits
- Machine Architecture

Lecture Outline (continued)

AUTOMATA and LANGUAGES

- REs and FSAs
- Grammars
- Chomsky Hierarchy
- Turing Machines

PROGRAMMING SYSTEMS

- Memory Organization
- Modules and Libraries
- Objects

ANALYSIS OF ALGORITHMS

- Algorithms and Complexity
- Improving Performance
- NP-completeness

APPLICATIONS

- Graphics
- Mathematical Applications
- Perspective

Precepts

Fridays: prepared topics

- getting online
- C basics
- TOY programs
- C structures
- C pointers
- Postscript
- grep/awk
- C libraries
- Java
- Typesetting

Mondays: discussion and questions

- reading
- exercises
- programming assignments

COS 126 Survival Guide

Keep up with the course materials

- read over handouts when you get them
- www.CS.Princeton.EDU/courses/archive/fall97/cs126
- prepare for precepts

Keep in touch

- finger, 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

Linear feedback shift register

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 is

1 if they are different

0 if they are the same

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

1 0 0 1 0 0 1 1 0 0 1

0 0 1 0 0 1 1 0 0 1 0

Bits "look" random (but aren’t!)

Using "Random" Bits for Encryption

Convert message to bitstream

```
    S     E     N     D     M     O     N     E     Y
  100100010101100001000110101110011000010111001
```

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

```
  100100010101100001000110101110011000010111001
  001001100100001101010100001111010100011100101
  101101110001101100010010100001001100001011100
```

Message looks random to anyone reading it

```
  101101110001101100010010100001001100001011100
    W     ?     M     R     E     A     F     B     Z
```

Receiver has identical machine

(Secretly) provide receiver with initial fill

Receiver computes XOR with SAME "random" bitstream

```
  101101110001101100010010100001001100001011100
  001001100100001101010100001111010100011100101
  100100010101100001000110101110011000010111001
    S     E     N     D     M     O     N     E     Y
```

Works because (a ^ b) ^ b = a ^ (b ^ b) = a

Properties of shift register "machine"

Clocked

Control: start, stop, or "load"
Data: initial values of bits

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;
         }
   }
```

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

Computer Systems and Abstract Machines

Layers of abstraction
- understand a machine, then
  - use it to build a more complex one
- substitute new (better) implementations
  - of abstract machines
- develop complex systems by building up

Evolution of "computer"
- Box with lights and switches
- Box with wires to keyboard and printer
- Box with wires to long-term memory (files),
  - keyboard, and terminal
- Bitmapped display with keyboard, mouse,
  - wires to file system

Programs transform input to output
Files can be input, output, or both

- Editor (emacs): create, modify files
- Compiler (cc): transform program from text
  - to machine instructions
- Operating System (Unix): invoke programs
- Windowing system (X) maintain the illusion of
  - multiple computer systems