Computer Science 126

General Computer Science
Fall 2014

Robert Sedgewick
Prologue:
A Simple Machine
Prologue: A Simple Machine

• Brief introduction
• Secure communication with a one-time pad
• Linear feedback shift registers
• Implications
Who are you?

[diagram showing intended major distribution with categories:
- Social Sciences
- other Science/Math
- other Engineering
- Humanities
- CS
- 1st year
- Sophomore
- Junior
- Senior]

Over 60% of all Princeton students take COS 126
What is this course about?

A broad introduction to computer science.

Goals
- Demystify computer systems.
- Empower you to exploit available technology.
- Build awareness of substantial intellectual underpinnings.

Topics
- Programming in Java.
- Design and architecture of computers.
- Theory of computation.
- Applications in science and engineering.

“Science is everything we understand well enough to explain to a computer.”
— Don Knuth

“Computers are incredibly fast, accurate, and stupid; humans are incredibly slow, inaccurate, and brilliant; together they are powerful beyond imagination.”
— Albert Einstein

and art, music, finance, and many other fields.
The basics

- **Lectures.** [Sedgewick]

- **RS office hours.** Everyone needs to meet me!

- **Precepts.** [Gabai, Ginsburg and team]
  - Tips on assignments / worked examples
  - Questions on lecture material.
  - Informal and interactive.

- **Friend 016/017 lab.** [undergraduate assistants]
  - Help with systems/debugging.
  - No help with course material.

- **Piazza.** [online discussion]
  - Best chance of quick response to a question.
  - Post to class or private post to staff.

See [www.princeton.edu/~cos126](http://www.princeton.edu/~cos126) for full current details and office hours.
Opportunities for us to determine your level of achievement:

- 9 programming assignments.
- 2 written exams (in class, 10/9 and 12/11).
- 2 programming exams (evenings, 10/23 and 12/8).
- Final programming project (due Dean's date – 1).
- Extra credit / staff discretion. Adjust borderline cases.

We do **not** grade on a "curve".

Grades are based on **achievement**.

Due dates:

<table>
<thead>
<tr>
<th></th>
<th>Su</th>
<th>Mo</th>
<th>Tu</th>
<th>We</th>
<th>Th</th>
<th>Fr</th>
<th>Sa</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEP</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>29</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCT</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAN</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Course Information

Course description. An introduction to computer science in the context of scientific, engineering, and commercial applications. The goal of the course is to teach basic principles and practical issues, while at the same time preparing students to use computers effectively for applications in computer science, physics, biology, chemistry, engineering, and other disciplines. Topics include: programming in Java; hardware and software systems; algorithms and data structures; fundamental principles of computation; and scientific computing, including simulation, optimization, and data analysis.

Instructor. Robert Sedgewick.

Lectures. Lectures meet on Tuesdays and Thursdays at 10am (L01)

Preceptors. Donna Gabai (co-lead) · Maia Ginsburg (co-lead) · Doug Clark · Andrea LaPaugh · Dan Leyzberg · Stephen Cook · Katie Edwards · Young Kun Ko · Theodore Brundage · Nevin Li · Jordan Ash · Shaoqing (Victor) Yang · Emily Nelson · Colin Watson

Precepts. Precepts meet twice a week on Tuesdays and Thursdays or Wednesdays and Fridays. Precepts begin either Thursday Sept 11 or Friday Sept 12.

Undergraduate coordinator. For enrollment problems, see Colleen Kenny-McGinley in CS 210.

Course website. The course website contains a wealth of information, including precept rosters, office hours, lecture slides, programming
Textbook and Booksite

Textbook.
- Full introduction to course material.
- Developed for this course.
- For use while learning and studying.

Booksite.
- Summary of content.
- Code, exercises, examples.
- Supplementary material.
- NOT the textbook.
- (also not the course web page).
- For use while online.

http://introcs.cs.princeton.edu

← bookmark this page, too!
Programming assignments
are an essential part of the experience in learning CS.

Desiderata
- Address an important scientific or commercial problem.
- Illustrate the importance of a fundamental CS concept.
- You solve the problem from scratch on your own computer!
What's Ahead?

**Coming events**
- Lecture 2. Basic programming concepts.
- Precept 1. Meets today/tomorrow.
- Not registered? Go to any precept now; officially register ASAP.
- Change precepts? Use SCORE. see Colleen Kenny-McGinley in CS 210 if the only precept you can attend is closed

**Assignment 0 due Monday 11:59PM**

**Things to do before attempting assignment**
- Read Sections 1.1 and 1.2 in textbook.
- Read assignment carefully.
- Install introcs software as per instructions.
- Do a few exercises.
- Lots of help available, don't be bashful.

http://introcs.cs.princeton.edu/ assignments.php

END OF ADMINISTRATIVE STUFF
1. Prologue: A Simple Machine

• Brief introduction
• **Secure communication with a one-time pad**
• Linear feedback shift registers
• Implications
Sending a secret message with a cryptographic key

Alice wants to send a secret message to Bob.
- Sometime in the past, they exchange a cryptographic key.
- Alice uses the key to encrypt the message.
- Bob uses the same key to decrypt the message.

Critical point: Without the key, Eve cannot understand the message.

Q. How does the system work?
Encrypt/decrypt methods

Goal. Design a method to encrypt and decrypt data.

Example 1. Enigma encryption machine [German code, WWII]
- Broken by Turing bombe (one of the first uses of a computer).
- Broken code helped win Battle of Atlantic by providing U-boat locations.

Example 2. One-time pad [details to follow]

Example 3. Linear feedback shift register [later this lecture]
A digital world

A *bit* is a basic unit of information.
- Two possible values (0 or 1).
- Easy to represent in the physical world (*on* or *off*).

In modern computing and communications systems, we represent *everything* as a sequence of bits.
- Text [details to follow in this lecture]
- Numbers
- Sound [details to follow in this course]
- Pictures [details to follow in this course]
- ...
- Programs [profound implications, stay tuned].

Bottom line. If we can send and receive bits, we can send and receive *anything*.

\[01001012 = 69_{10}\]
Encoding text as a sequence of bits

Base64 encoding of character strings

- A simple method for representing text.
- 64 different symbols allowed: A–Z, a–z, 0–9, +, /.
- 6 bits to represent each symbol.
- ASCII and Unicode methods used on your computer are similar.

<table>
<thead>
<tr>
<th></th>
<th>bits</th>
<th>symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base64</td>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>ASCII</td>
<td>8</td>
<td>256</td>
</tr>
<tr>
<td>Unicode</td>
<td>16</td>
<td>65,536+</td>
</tr>
</tbody>
</table>

Example:

SEND MONEY

SEND MONE Y

100100100010001110100001110010011000011010001101010010010011000
One-Time Pads

What is a one-time pad?

• A cryptographic key known only to the sender and receiver.
• Good choice: A random sequence of bits (stay tuned).
• Security depends on each sequence being used only once.

Note: Any sequence of bits can be decoded into a sequence of characters.
Encryption with a one-time pad

**Preparation**
- Create a "random" sequence of bits (a one-time pad).
- Send one-time pad to intended recipient through a secure channel.

**Encryption**
- Encode text as a sequence of $N$ bits.
- Use the first $N$ bits of the pad.
- Compute a new sequence of $N$ bits from the message and the pad.
- Decode result to get a sequence of characters.

Result: A ciphertext (encrypted message).

---

**Example**

**Message:** SENDMONEY

**One-time pad:** yT25a5i/S

**Ciphertext:** gX76W3v7K

---

Important point: need to have as many bits in the pad as there are in the message.
A (very) simple machine for encryption

To compute a cyphertext from a message and a one-time pad
- Encode message and pad in binary.
- Each cyphertext bit is the \textit{bitwise exclusive or} of corresponding bits in message and pad.

\textbf{Def.} The \textit{bitwise exclusive or} of two bits is 1 if they differ, 0 if they are the same.
Self-assessment on bitwise XOR encryption

Q. Encrypt the message E A S Y with the pad 0 1 2 3.
Decryption with a one-time pad

Sending a secret message with a cryptographic key

Alice wants to send a secret message to Bob.
- Sometime in the past, they exchange a cryptographic key.
- Alice uses the key to encrypt the message.
- Bob uses the *same* key to decrypt the message.

Critical point: Without the key, Eve cannot understand the message.

**Q.** How does the system work?

**A.** Alice's device uses a "bitwise exclusive or" machine to encrypt the message.

**Q.** What kind of machine does Bob's device use to *decrypt* the message?

**A.** The same one (!!)

"use yT25a5J/s if I ever send you an encrypted message" (Bob, to Alice)

"Ok!" (Alice, to Bob)

```
Hey, Bob. Here's a secret message.
key: yT25a5J/s
SENDMONEY sending gX76W3v7K
```

```
Hey, Bob. Here's a secret message.
key: yT25a5J/s
SENDMONEY
```

encrypted message is "in the clear" (anyone can read it)
A (very) simple machine for encryption and decryption

To compute a message from a cyphertext and a one-time pad
- Use binary encoding of cyphertext and pad.
- Each message bit is the bitwise exclusive or of corresponding bits in cyphertext and pad.

```
| g X 7 6 w 3 v 7 K | cyphertext | 10000001011111010101101101111101111101010 |
| y T 2 5 a 5 i / S | one-time pad | 1101001001111011011010111001100011000110011001111101010 |
| SENDMONEY         | message     | 01001000010001101000011001100011100110011010010010011000 |
```

1 if they differ; 0 if they are the same
**Why does it work?**

**Crucial property:** Decrypted message is the same as the original message.
Let $m$ be a bit of the message and $k$ be the corresponding bit of the one-time pad.

To prove: \((m \land k) \land k = m\)

**Approach 1: Truth tables**

<table>
<thead>
<tr>
<th>$m$</th>
<th>$k$</th>
<th>$m \land k$</th>
<th>$(m \land k) \land k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Approach 2: Boolean algebra**

\[
(k \land k) = 0
\]
\[
m \land 0 = m
\]
\[
(m \land k) \land k = m \land (k \land k)
\]
\[
= m \land 0
\]
\[
= m
\]
Decryption with the wrong pad

Eve cannot read a message without knowing the pad.

One-time pad is provably secure [Shannon, 1940s]
- IF each pad is used only once,
- AND the pad bits are random,
- THEN Eve cannot distinguish cyphertext from random bits.
Eve's problem with one-time pads

Eve has a computer. Why not try all possibilities?

Problem
• 54 bits, so there are $2^{54}$ possible pad values.
• Suppose Eve could check a million values per second.
• It would still take 570+ years to check all possibilities.

Much worse problem
• There are also $2^{54}$ possible messages.
• If Eve were to check all the pads, she'd see all the messages.
• No way to distinguish the real one from any other.

One-time pad is provably secure.
Goods and bads of one-time pads

**Goods.**
- Very simple encryption method.
- Decrypt with the same method.
- Provably unbreakable if bits are truly random.
- Widely used in practice.

**Bads.**
- Easily breakable if seed is re-used.
- Truly random bits are very hard to come by.
- Need separate secure channel to distribute key.
- Pad must be as long as the message.
Random bits are not so easy to find

You might look on the internet.

The randomness comes from atmospheric noise

"I think I'll call it random.org"

... if you trust the internet.

Next: Creating a (long) sequence of "pseudo-random" bits from a (short) key.
Prologue: A Simple Machine

- Brief introduction
- Secure communication with a one-time pad
- Linear feedback shift registers
- Implications
A pseudo-random number generator

is a *deterministic* machine that produces a long sequence of *pseudo random* bits.

**Examples**

- Enigma.
- Linear feedback shift register (next).
- Blum-Blum-Shub generator.
- ...
- [ an early application of computing ]
- [ research still ongoing ]

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

– John von Neumann
A pseudo-random number generator

is a *deterministic* machine that produces a long sequence of *pseudo random* bits.

**Deterministic:** Given the current state of the machine, we know the next bit.

**Random:** We never know the next bit.

**Pseudo-random:** The sequence of bits *appears to be random.*

*Appears to be random??*

- A profound and elusive concept.
- For this lecture: "Has enough properties of a random sequence that Eve can't tell the difference".

Ex. 1: No long repeats
Ex. 2: About the same number of 0s and 1s
Ex. 3: About the same number of 00s, 01s, 10s, and 11s.
...
Which of these sequences appear to be random?

Note: Any one of them could be random!
### Linear feedback shift register

#### Terminology

- **Bit**: 0 or 1.
- **Cell**: storage element that holds one bit.
- **Register**: sequence of cells.
- **Seed**: initial sequence of bits.
- **Feedback**: Compute XOR of two bits and put result at right.
- **Shift register**: when clock ticks, bits propagate one position to left.

#### More terminology

- **Tap**: Bit positions used for XOR (one must be leftmost).
- **\([N, k]\) LFSR**: \(N\)-bit register with taps at \(N\) and \(k\).
Linear feedback shift register simulation

History of register contents

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0 1 1 0 1 0 0 0 0 1 0 1</td>
<td>1 1 0 1 0 0 0 0 1 0 1</td>
<td>1 0 1 0 0 0 0 1 0 1 1</td>
<td>0 1 0 0 0 0 1 0 1 1 0</td>
<td>1 0 0 0 0 1 0 1 1 0 0</td>
<td>0 0 0 0 1 0 1 1 0 0 1</td>
</tr>
<tr>
<td>Y</td>
<td>0 1 1 0 1 0 0 0 0 1 0 1</td>
<td>1 1 0 1 0 0 0 0 1 0 1</td>
<td>1 0 1 0 0 0 0 1 0 1 1</td>
<td>0 1 0 0 0 0 1 0 1 1 0</td>
<td>1 0 0 0 0 1 0 1 1 0 0</td>
<td>0 0 0 0 1 0 1 1 0 0 1</td>
</tr>
<tr>
<td>W</td>
<td>0 1 1 0 1 0 0 0 0 1 0 1</td>
<td>1 1 0 1 0 0 0 0 1 0 1</td>
<td>1 0 1 0 0 0 0 1 0 1 1</td>
<td>0 1 0 0 0 0 1 0 1 1 0</td>
<td>1 0 0 0 0 1 0 1 1 0 0</td>
<td>0 0 0 0 1 0 1 1 0 0 1</td>
</tr>
</tbody>
</table>

a pseudo-random bit sequence!
A random bit sequence?

Q. Is this a random sequence?

Looks random to me.

No long repeats.
997 0s, 1003 1s.
256 00s, 254 01s, 256 10s, 257 11s.
...

one-time pad in our example

A. No. It is the output of an [11, 9] LFSR with seed 01101000010!

It is pseudo-random
(at least to some observers.)
Self-assessment on LFSRs

Q. Give first 10 steps of [5,4] LFSR with initial fill 00001.
Encryption/decryption with an LFSR

**Preparation**
- Alice creates a book of "random" (short) seeds.
- Alice sends the **book** to Bob through a secure channel.

**Encryption/decryption**
- Alice sends Bob a description of which seed to use.
- They use the specified seed to initialize an LFSR and produce \( N \) bits. [and proceed in the same way as for one-time pads]

```
message  SENDMONEY  010010000100001101000011001100001110001101000100011000
seed      01101000010  LFSR  110010010011110110111001011010111001100010111111010010
```

```
message  SENDMONEY  010010000100001101000011001100001110001101000100011000
seed      01101000010  LFSR  110010010011110110111001011010111001100010111111010010
```

```
message  SENDMONEY  010010000100001101000011001100001110001101000100011000
seed      01101000010  LFSR  110010010011110110111001011010111001100010111111010010
```

```
message  SENDMONEY  010010000100001101000011001100001110001101000100011000
seed      01101000010  LFSR  110010010011110110111001011010111001100010111111010010
```
Eve's opportunity with LFSR encryption

Eve has computers. Why not try all possible seeds?
- Seeds are short, messages are long.
- All seeds give a tiny fraction of all messages.
- Extremely likely that all but real seed will produce gibberish.

Good news (for Eve): This approach can work.
- Ex: 11-bit register implies 2047 possibilities.
- Extremely likely that only one of those is not gibberish.
- After this course, you could write a program to check whether any of the 2047 messages have words in the dictionary.

Bad news (for Eve): It is easy for Alice and Bob to use a much longer LFSR.
Key properties of LFSRs

Property 1.
• Don’t use all 0s as a seed!
• Fill of all 0s will not otherwise occur.
Key properties of LFSRs

**Property 1.**
- Don’t use all 0s as a seed!
- Fill of all 0s will not otherwise occur.

**Property 2.** Bitstream must eventually cycle.
- \(2^N - 1\) nonzero fills in an \(N\)-bit register.
- Future output completely determined by current fill.

Ex. [4,3] LFSR

```
0 0 1 0 0 0
0 1 0 0 1 1
1 0 0 1 1 2
0 0 1 1 0 3
0 1 1 0 1 4
1 1 0 1 0 5
1 0 1 0 1 6
0 1 0 1 1 7
1 0 1 1 1 8
0 1 1 1 1 9
1 1 1 1 0 10
1 1 1 0 0 11
1 1 0 0 0 12
1 0 0 0 1 13
0 0 0 1 0 14
0 0 1 0 15
```
Key properties of LFSRs

Property 1.
• Don’t use all 0s as a seed!
• Fill of all 0s will not otherwise occur.

Property 2. Bitstream must eventually cycle.
• $2^N - 1$ nonzero fills in an $N$-bit register.
• Future output completely determined by current fill.

Property 3. Cycle length in an $N$-bit register is \emph{at most} $2^N - 1$.
• Could be smaller; cycle length depends on tap positions.
• Need theory of finite groups to know good tap positions.
Key properties of LFSRs

Property 1.
- Don’t use all 0s as a seed!
- Fill of all 0s will not otherwise occur.

Property 2. Bitstream must eventually cycle.
- \(2^N - 1\) nonzero fills in an \(N\)-bit register.
- Future output completely determined by current fill.

Property 3. Cycle length in an \(N\)-bit register is at most \(2^N - 1\).
- Could be smaller; cycle length depends on tap positions.
- Need theory of finite groups to know good tap positions.

Bottom line.
- [63, 62] register generates \(2^{63} - 1\) bits before repeating.

XILINX manual, 1990s

Definitely preferable: small cost, huge payoff.
Eve's problem with LFSR encryption

Without the seed, Eve cannot read the message.

Eve has computers. Why not try all possible seeds?
• Seeds are short, messages are long.
• All seeds give a tiny fraction of all messages.
• Extremely likely that all but real seed will produce gibberish.

Bad news (for Eve): There are still way too many possibilities.
• Ex: 63-bit register implies $2^{63} - 1$ possibilities.
• If Eve could check 1 million seeds per second, it would take her 2923 centuries to try them all!

Bad news (for Alice and Bob): LFSR output is not random.
Goods and bads of LFSRs

**Goods.**
- Very simple encryption method.
- Decrypt with the same method.
- Scalable: 20 cells for 1 million bits; 30 cells for 1 billion bits.
- Widely used in practice. [Example: military cryptosystems.]

**Bads.**
- Easily breakable if seed is re-used.
- Still need secure key distribution.
- Experts can crack LFSR encryption.

**Example.**
- CSS encryption widely used for DVDs.
- Widely available DeCSS breaks it!
Prologue: A Simple Machine

- Brief introduction
- Secure communication with a one-time pad
- Linear feedback shift registers
- Implications
LFSRs and general-purpose computers

Important similarities.
- Both are built from simple components.
- Both scale to handle huge problems.
- Both require careful study to use effectively.

<table>
<thead>
<tr>
<th>component</th>
<th>LFSR</th>
<th>computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>control</td>
<td>start, stop, load</td>
<td>same</td>
</tr>
<tr>
<td>clock</td>
<td></td>
<td>same</td>
</tr>
<tr>
<td>memory</td>
<td>12 bits</td>
<td>billions of bits</td>
</tr>
<tr>
<td>input</td>
<td>12 bits</td>
<td>bit sequence</td>
</tr>
<tr>
<td>computation</td>
<td>shift, XOR</td>
<td>+ – * / ...</td>
</tr>
<tr>
<td>output</td>
<td>pseudo-random bit sequence</td>
<td>any computable bit sequence</td>
</tr>
</tbody>
</table>

Critical differences: Operations, input. but the simplest computers differ only slightly from LFSRs!
- General purpose computer can simulate any abstract machine.
- All general purpose computers have equivalent power (!) [stay tuned].
A Profound Idea

Programming. We can write a Java program to simulate the operation of any abstract machine.
- Basis for theoretical understanding of computation.
- Basis for bootstrapping real machines into existence.
Stay tuned (we cover these sorts of issues in this course).

```
public class LFSR {
    public static void main(String[] args) {
        int[] a = { 0, 0, 1, 0, 0, 0, 1, 0, 1, 0 }; // Set the initial state
        for (int t = 0; t < 2000; t++) {
            a[0] = (a[11] ^ a[9]); // Apply the operation
            System.out.print(a[0]); // Print the current bit
            for (int i = 11; i > 0; i--) { // Shift the state
                a[i] = a[i-1];
            }
            System.out.println();
        }
    }
}
```

Note: You will write and apply an LFSR simulator in Assignment 5.
Profound questions

**Q.** What is a random number? 

LFSRs *do not* produce random numbers.

- They are *deterministic.* ~~~~ von Neumann's "state of sin": we know that "deterministic" is incompatible with "random"
- It is not obvious how to distinguish the bits LFSRs produce from random,
- BUT experts have figured out how to do so.

**Q.** Are random processes found in nature?
- Motion of cosmic rays or subatomic particles?
- Mutations in DNA?

**Q.** Is the natural world a (not-so-simple) deterministic machine??

"God does not play dice."

— Albert Einstein
1. Prologue:
A Simple Machine