ASSIGNMENT 6 TIPS AND TRICKS

- digital audio review
- guitar string data type
- ring buffer data type
- guitar hero client

http://princeton.edu/~cos126
Goals

- Physically-modeled sound: compute sound waveform using a mathematical model of a musical instrument.
Goals

- Physically-modeled sound: compute sound waveform using a mathematical model of a musical instrument.
- Object-oriented programming: more practice with objects.
- Performance: efficient data structure crucial for application.
ASSIGNMENT 6 TIPS AND TRICKS

- digital audio review
- guitar string data type
- ring buffer data type
- guitar hero client
**Waveform.** Real-valued function between −1 and +1.

\[ a(t) = \sin (2\pi \cdot t \cdot 440), \quad 0 \leq t \leq T \]

**Pure tone.** Periodic sinusoidal waveform.
Digital sound

**Digital representation.** Sample at equally-spaced points.

5,512 samples per second (138 samples)

$T = \frac{1}{40 \text{ second}}$
Digital sound

**Digital representation.** Sample at equally-spaced points.

11,025 samples per second (276 samples)

$T = 1/40 \text{ second}$
Digital sound

Digital representation. Sample at equally-spaced points.

22,050 samples per second (552 samples)

T = 1/40 second
Digital sound

Digital representation. Sample at equally-spaced points.

44,100 samples per second (1,103 samples)

T = 1/40 second
Digital sound

Digital representation. Sample at equally-spaced points.

\[ a[i] = \sin \left( \frac{2\pi \cdot i \cdot 440}{44100} \right), \quad i = 0, 1, 2, \ldots, 44100 \cdot T \]

for (int i = 0; i <= 44100 * T; i++) {
    double x = Math.sin(2.0 * Math.PI * i * 440.0 / 44100);
    StdAudio.play(x);
}
Digital sound

**Digital representation.** Sample at equally-spaced points.

### 44,100 samples per second (1,103 samples)

```java
for (int i = 0; i <= 44100 * T; i++) {
    double x = Math.sin(2.0 * Math.PI * i * 17000.0 / 44100);
    StdAudio.play(x);
}
```

**Ringtone torture.**

\[ T = \frac{1}{40} \text{ second} \]
Real-time audio library

**Standard audio.** Simple library to play sound in Java.

- User sends samples to standard audio.
- Standard audio sends them to sound card at 44,100 Hz.

```java
public class StdAudio {

    public static int SAMPLE_RATE = 44,100 (CD-quality audio)

    public static void play(double x) { write one sample to sound card }

    public static void play(double[] x) { write array of samples to sound card }

    public static double[] read(String filename) { read audio samples from wav file }

    public static void save(...) { save audio samples to wav file }
}
```
ASSIGNMENT 6 TIPS AND TRICKS

- digital audio review
- guitar string data type
- ring buffer data type
- guitar hero client
Transverse wave demo
Longitudinal wave demo

(a)

(b)

(c)
Modeling the guitar string

Physical guitar string.
- Length of string determines fundamental frequency.†
- Once plucked, string vibrates.
- Amplitude decreases as energy dissipates into sound and heat.

Digital model. Sequence of $n$ displacements, where $n = \lceil 44.100 / \text{frequency} \rceil$. 

Math.ceil()
Modeling the plucking of a guitar string

Plucking a guitar string. Excitation can contain energy at any frequency.

White noise. Set each of $n$ displacements uniform at random in $(-\frac{1}{2}, \frac{1}{2})$. 
Simulating the vibrating guitar string: Karplus–Strong

Karplus.

- Play the first sample.
- Peek at first two samples (and remove first).
- Append the average of those two samples, scaled by an energy dissipation factor of 0.996.

\[
\text{before} \quad \begin{array}{cccccccc}
.2 & .4 & .5 & .3 & -.2 & .4 & .3 & .0 & -.1 & -.3 \\
\end{array}
\]

\[
.996 \times \frac{1}{2} ( .2 + .4 )
\]

\[
\text{after} \quad \begin{array}{cccccccc}
.2 & .4 & .5 & .3 & -.2 & .4 & .3 & .0 & -.1 & -.3 & .2988 \\
\end{array}
\]

Strong. Sampling the transversal wave on a string instrument.
public class GuitarString

public GuitarString(double freq)  creates a guitar string of given frequency
public GuitarString(double[] init) for unit testing
public int length() returns the length of this guitar string
public void pluck() plucks this guitar string
public void tic() advances the simulation one time step
public double sample() returns the current sample

GuitarString concertA = new GuitarString(440.0);
concertA.pluck();
while (true) {
    StdAudio.play(concertA.sample());
    concertA.tic();
}
Guitar string implementation

Q. How to represent?
A. Need data structure that can remove value from front and add to back.

Core operations needed.
- **Construct**: create a data structure (capable of holding \( n \) items).
- **Enqueue**: add value.
- **Dequeue**: remove least recently added value.
- **Peek**: look at least recently added value.

```
before  .2  .4  .5  .3  -.2  .4  .3  .0  -.1  -.3

.996 \times \frac{1}{2} ( .2 + .4 )

after  .2  .4  .5  .3  -.2  .4  .3  .0  -.1  -.3  .2988
```
ASSIGNMENT 6 TIPS AND TRICKS

- digital audio review
- guitar string data type
- ring buffer data type
- guitar hero client
Ring buffer API

**Goal.** Design a data type that can implement Karplus–Strong.

```java
public class RingBuffer
```

- `public RingBuffer(int capacity)`  
  *creates an empty ring buffer of given capacity*

- `public int capacity()`  
  *maximum number of items in buffer*

- `public int size()`  
  *number of items currently in buffer*

- `public boolean isEmpty()`  
  *is this ring buffer empty?*

- `public boolean isFull()`  
  *is this ring buffer full?*

- `public void enqueue(double x)`  
  *adds item x to the end*

- `public double dequeue()`  
  *removes and returns item from front*

- `public double peek()`  
  *returns item from front*

**Performance requirement.** All instance methods must take *constant time* (called 44,100 times per second).
Ring buffer implementation

Performance bug.
- Enqueue: add item at $a[n]$ and increment $n$. 

```
enqueue  9
```

```
a[]    0 1 2 3 4 5 6 7 8 9
             n
```

3 1 4 1 5
Ring buffer implementation

Performance bug.

- Enqueue: add item at $a[n]$ and increment $n$.  

```c
const int a[] = {3, 1, 4, 1, 5, 9, 0, 1, 2, 3};
```

enqueue

- Constant time per op
Ring buffer implementation

Performance bug.
- Enqueue: add item at $a[n]$ and increment $n$.
- Dequeue: remove item $a[0]$ and shift all items.

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
a[] & 3 & 1 & 4 & 1 & 5 & 9 & \_ & \_ & \_ \\
\end{array}
\]

\[\text{--- constant time per op} \]
Ring buffer implementation

**Performance bug.**

- Enqueue: add item at $a[n]$ and increment $n$.
- Dequeue: remove item $a[0]$ and shift all items.

```
   dequeue  3
```

<table>
<thead>
<tr>
<th>a[]</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\leftarrow$ constant time per op

$\leftarrow$ linear time per op
Ring buffer implementation

Performance bug.
- Enqueue: add item at a[n] and increment n.
- Dequeue: remove item a[0] and shift all items.

Bottom line. Too slow to generate samples at 44.1kHz!
**Ring buffer implementation**

**Efficient implementation.**

- Enqueue: add item at \(a[\text{last}]\) and increment \(\text{last}\).
Ring buffer implementation

Efficient implementation.

- Enqueue: add item at \(a[\text{last}]\) and increment \(\text{last}\).  
  
  ![enqueue diagram]

\[\text{enqueue}\]

\[a[\text{last}]\]

\[0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9\]

\([0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]\]

\[\text{first} \quad \text{last}\]

Constant time per op
Ring buffer implementation

Efficient implementation.

- Enqueue: add item at a[last] and increment last.
- Dequeue: remove item a[first] and increment first.

\[\text{dequeue}\]

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

first \hspace*{2cm} last

constant time per op
Ring buffer implementation

Efficient implementation.

- Enqueue: add item at \[a[\text{last}]\] and increment \(\text{last}\).
- Dequeue: remove item \(a[\text{first}]\) and increment \(\text{first}\).  
  
\[\text{constant time per op}\]

\[\text{dequeue} \quad 3\]

\[
\begin{array}{ccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
\text{a[]} & & 1 & 4 & 1 & 5 & 9 & & & \\
\end{array}
\]

\(\text{first} \quad \text{last}\)
Ring buffer implementation

Efficient implementation.

- Enqueue: add item at a[last] and increment last.
- Dequeue: remove item a[first] and increment first.

enqueue 2

\[
\begin{array}{ccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
\end{array}
\]

a[] 1 4 1 5 9

first last

constant time per op
Ring buffer implementation

Efficient implementation.

- Enqueue: add item at \(a[\text{last}]\) and increment \(\text{last}\).  \[
\text{constant time per op}
\]
- Dequeue: remove item \(a[\text{first}]\) and increment \(\text{first}\).  \[
\text{constant time per op}
\]

\[
\begin{array}{c|cccccccccc|}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
a[] & [ & [ & 1 & 4 & 1 & 5 & 9 & 2 & ] & ] \\
\hline
\text{first} & \text{last}
\end{array}
\]
Ring buffer implementation

Efficient implementation.

- Enqueue: add item at \(a[\text{last}]\) and increment \(\text{last}\).
- Dequeue: remove item \(a[\text{first}]\) and increment \(\text{first}\).

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
\text{first} & & & & 1 & 4 & 1 & 5 & 9 & 2 \\
\text{last} & & & & & & & & & \\
\end{array}
\]

\underline{enqueue} 6
Ring buffer implementation

Efficient implementation.

- Enqueue: add item at \( a[\text{last}] \) and increment \( \text{last} \).
- Dequeue: remove item \( a[\text{first}] \) and increment \( \text{first} \).

\[ \begin{array}{ccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
a[] & \text{first} & \text{last} & 1 & 4 & 1 & 5 & 9 & 2 & 6
\end{array} \]

\[ \text{constant time per op} \]
Efficient implementation.

- Enqueue: add item at $a[\text{last}]$ and increment $\text{last}$.  \hfill \text{constant time per op}
- Dequeue: remove item $a[\text{first}]$ and increment $\text{first}$.  \hfill \text{constant time per op}
- Use cyclic wrap-around (compute indices modulo capacity).

**enqueue** 5

\begin{center}
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
ap[] & \_ & \_ & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
\text{last} & \text{first}
\end{array}
\end{center}
Ring buffer implementation

Efficient implementation.

- Enqueue: add item at a[last] and increment last.
- Dequeue: remove item a[first] and increment first.
- Use cyclic wrap-around (compute indices modulo capacity).

```
<table>
<thead>
<tr>
<th>enqueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[]</td>
</tr>
<tr>
<td>5 1 4 1 5 9 2 6</td>
</tr>
<tr>
<td>last</td>
</tr>
<tr>
<td>first</td>
</tr>
</tbody>
</table>
```
Ring buffer implementation: performance matters

Q. I have a quad-core MacBook Pro with 16GB memory.

Does constant time vs. linear time matter in practice?

A. Yes!

concert A (efficient implementation)

concert A (performance bug)

Remark. Could use same trick to speed up LFSR.
public class RingBuffer {
    private double[] a;    // elements
    private int first;     // index of dequeue element
    private int last;      // index of enqueue element

    public int size() {
        // YOUR CODE HERE
    } // YOUR CODE HERE

    ...}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>.5</th>
<th>.3</th>
<th>-.2</th>
<th>.4</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

    first     last     capacity
public class RingBuffer {
    private double[] a; // elements
    private int first; // index of dequeue element
    private int last; // index of enqueue element
    private int capacity // number of elements

    public int size() {
        return last - first;
    }
    ...
}

why wrong?
Assignment 6 Tips and Tricks

- digital audio review
- guitar string data type
- ring buffer data type
- guitar hero client
A 1-string guitar

```java
public class GuitarHeroUltraLite {
    public static void main(String[] args) {
        GuitarString stringA = new GuitarString(440.0);  // concert A

        while (true) {
            if (StdDraw.hasNextKeyTyped()) {
                char key = StdDraw.nextKeyTyped();
                if (key == 'a') stringA.pluck();
            }

            StdAudio.play(stringA.sample());  // play the sample
            stringA.tic();  // do Karplus–Strong update
        }
    }
}
```
A 37-string guitar

Model many simultaneously vibrating guitar strings.

- Classic guitar has 6 strings and 19 frets.
- Our digital guitar has 37 strings.
- Create an array of GuitarString objects.
- Apply law of superposition.

\[
\text{string } i \text{ has frequency } 440 \times 2^{(i-24)/12}
\]

A major chord

\begin{align*}
A & \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim 440.00 \\
C# & \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim 554.37 \\
E & \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim 659.26
\end{align*}
A 37-string guitar

Model many simultaneously vibrating guitar strings.

- Classic guitar has 6 strings and 19 frets.
- Our digital guitar has 37 strings.
- Create an array of GuitarString objects.
- Apply law of superposition.

\[
\text{string } i \text{ has frequency } 440 \times 2^{(i-24)/12}
\]

A major chord

A major
Making a musical instrument

User interface. User types key to pluck string.

A scale: i o - [ z d f v
Stairway to Heaven
Modeling the 37 strings

How to map from a keystroke to corresponding GuitarString object?

A. 37-way if statement  
B. 37-way switch statement  
C. an array/string of 37 characters  
D. a symbol table with char keys and GuitarString values

don't even think about it!
good idea, but symbol tables not yet introduced in course

String keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
...
keyboard.length();  // 37 (don't hardwire 37!)
keyboard.indexOf('q');  // 0
keyboard.indexOf('r');  // 5
keyboard.indexOf('+');  // -1