CS Data Structures
Mr. Bredemeier

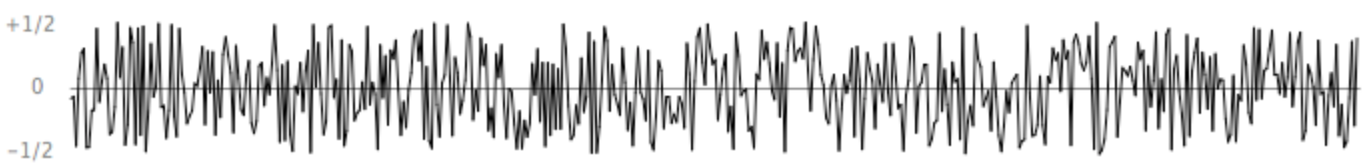
Guitar Heroine

This assignment was originally developed by Kevin Wayne at Princeton University (here is the link to the original assignment).

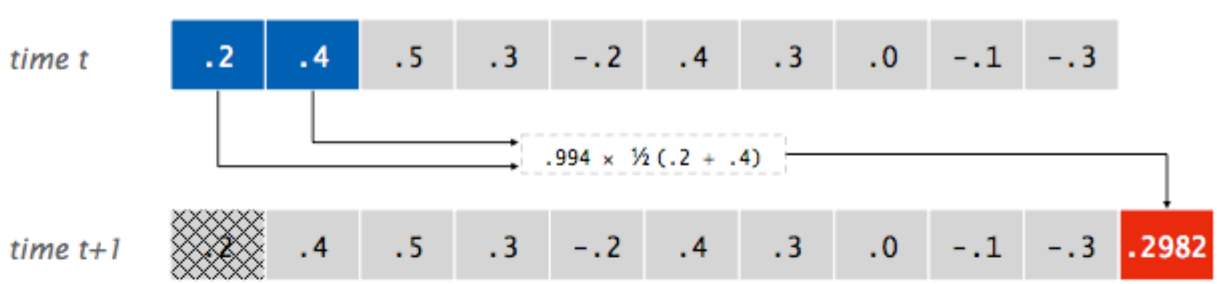
Write a program to simulate plucking a guitar string using the *Karplus-Strong* algorithm. This algorithm played a seminal role in the emergence of physically modeled sound synthesis (where a physical description of a musical instrument is used to synthesize sound electronically).

Simulate the plucking of a guitar string. When a guitar string is plucked, the string vibrates and creates sound. The length of the string determines its *fundamental frequency* of vibration. We model a guitar string by sampling its *displacement* (a real number between -1/2 and +1/2) at *N* equally spaced points (in time), where *N* equals the *sampling rate* (44,100) divided by the fundamental frequency (rounding the quotient up to the nearest integer).

- *Plucking the string.* The excitation of the string can contain energy at any frequency. We simulate the excitation with *white noise*: set each of the *N* displacements to a random real number between -1/2 and +1/2.



- *The resulting vibrations.* After the string is plucked, the string vibrates. The pluck causes a displacement which spreads wave-like over time. The Karplus-Strong algorithm simulates this vibration by maintaining a *ring buffer* of the *N* samples: the algorithm repeatedly deletes the first sample from the buffer and adds to the end of the buffer the average of the first two samples, scaled by an *energy decay factor* of 0.994.



the Karplus-Strong update

Why it works? The two primary components that make the Karplus-Strong algorithm work are the ring buffer feedback mechanism and the averaging operation.

- *The ring buffer feedback mechanism.* The ring buffer models the medium (a string tied down at both ends) in which the energy travels back and forth. The length of the ring buffer determines the fundamental frequency of the resulting sound. Sonically, the feedback mechanism reinforces only the fundamental frequency and its harmonics (frequencies at integer multiples of the fundamental). The energy decay factor (.994 in this case) models the slight dissipation in energy as the wave makes a roundtrip through the string.
- *The averaging operation.* The averaging operation serves as a gentle *low-pass filter* (which removes higher frequencies while allowing lower frequencies to pass, hence the name). Because it is in the path of the feedback, this has the effect of gradually attenuating the higher harmonics while keeping the lower ones, which corresponds closely with how a plucked guitar string sounds.

From a mathematical physics viewpoint, the Karplus-Strong algorithm approximately solves the 1D wave equation, which describes the transverse motion of the string as a function of time.

Ring buffer. Your first task is to create a data type to model the ring buffer. Complete the `RingBuffer` class that implements the following API:

```
public class RingBuffer
-------------------------------------------------------------------------
        RingBuffer(int capacity)  // create an empty ring buffer,
                                  // with given capacity
    int size()                    // return number of items in buffer
boolean isEmpty()                 // is the buffer empty
boolean isFull()                  // is the buffer full
   void add(double x)             // add item x to the end
 double peek()                    // return (but do not delete) item
                                  // from the front
 double remove()                  // delete and return item from the
                                  // front
```

Since the ring buffer has a known maximum capacity, implement it using a `double` array of that length. For efficiency, use *cyclic wrap-around*: Maintain one integer instance variable `front` that stores the index of the least recently inserted item; maintain a second integer instance variable `rear` that stores the index one beyond the most recently inserted item. To insert an item, put it at index `rear` and increment `rear`. To remove an item, take it from index `front` and increment `front`. When either index equals `capacity`, make it wrap-around by changing the index to `0`.

You should test your `RingBuffer` data type by invoking the provided `main` method. It will `add` the numbers 1 through 100, and then repeatedly `remove` the first two, and `add` their sum. If it is working properly, you will get the following results.

```
Size after wrap-around is 100
5050.0
```

Make sure `RingBuffer` is working properly before proceeding any further.

Guitar string. Next, create a data type to model a vibrating guitar string. Complete the `GuitarString` class that implements the following API:

```
public class GuitarString
--------------------------------------------------------------------------
       GuitarString(double frequency)  // create a guitar string of the
                                       // given frequency, using a
                                       // sampling rate of 44,100
       GuitarString(double[] array)    // create a guitar string whose
                                       // size and initial values are
                                       // given by the array
  void pluck()                         // set the buffer to white noise
  void tic()                           // advance one time step
double sample()                        // return the current sample
   int time()                          // return number of tics
```

- *Constructors.* There are two ways to create a `GuitarString` object.
  - The first constructor creates a `RingBuffer` of the desired capacity *N* (sampling rate 44,100 divided by *frequency*, rounded up to the nearest integer), and initializes it to represent a guitar string at rest by enqueueing *N* zeros.
  - The second constructor creates a `RingBuffer` of capacity equal to the size of the array, and initializes the contents of the buffer to the values in the array. On this assignment, its main purpose is for debugging and grading.
- *Pluck.* Remove and replace the *N* items in the ring buffer with *N* random values between -0.5 and +0.5.
- *Tic.* Apply the Karplus-Strong update: delete the sample at the front of the ring buffer and add to the end of the ring buffer the average of the first two samples, multiplied by the energy decay factor.
- *Sample.* Return the value of the item at the front of the ring buffer.
- *Time.* Return the total number of times `tic()` was called.

You should test your `GuitarString` data type by invoking the provided `main` method. It will create a `GuitarString` object and load it with an array of values, and display some of the string vibration iterations. If it is working properly, you will get the following results.

```
     0   0.2000                       13   0.0497
     1   0.4000                       14   0.0994
     2   0.5000                       15   0.3479
     3   0.3000                       16   0.1491
     4  -0.2000                       17  -0.0497
     5   0.4000                       18  -0.1988
     6   0.3000                       19  -0.0009
     7   0.0000                       20   0.3705
     8  -0.1000                       21   0.4199
     9  -0.3000                       22   0.2223
    10   0.2982                       23   0.0741
    11   0.4473                       24   0.2223
    12   0.3976
```

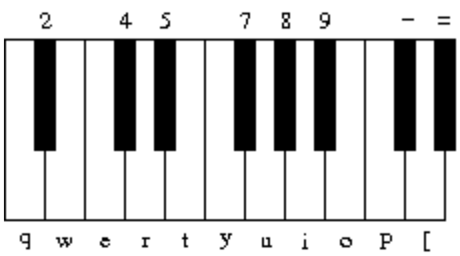
Make sure `GuitarString` is working properly before proceeding any further.

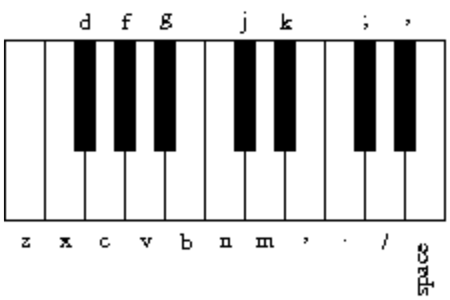
Interactive guitar player. `GuitarHeroineLite` is a sample `GuitarString` client that plays the guitar in real-time, using the keyboard to input notes. When the user types the lowercase letter `'a'` or `'c'`, the program plucks the corresponding string. Since the combined result of several sound waves is the superposition of the individual sound waves, we play the sum of all string samples.

Run `GuitarHeroineLite` to make sure that you can hear the two strings play clearly on your computer speakers.

Then, complete the assignment by writing a `GuitarHeroine` class that is similar to `GuitarHeroineLite`, but supports a total of 37 notes on the chromatic scale from 110Hz to 880Hz. In general, make the *i*th character of the string below play the *i* note.

```
String keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
```





This keyboard arrangement imitates a piano keyboard: The "white keys" are on the `qwerty` and `zxcv` rows and the "black keys" on the `12345` and `asdf` rows of the keyboard.

The *i*th character of the string corresponds to a frequency of $440 \times 1.05956^{(i-24)}$, so that the character `'q'` is approximately 110Hz, `'i'` is close to 220Hz, `'v'` is close to 440Hz, and `' '` is close to 880Hz. Create an array of 37 `GuitarString` objects and use `keyboard.indexOf(key)` to figure out which key was typed. Make sure your program does not crash if a key is played that is not one of your 37 notes.

Remember that the combined result of several sound waves is the superposition of the individual sound waves, so you will want to be sure to play the sum of *all* the string samples.

Finally, demonstrate your work by entering the following keystrokes into your guitar to get the beginning of Led Zeppelin's *Stairway to Heaven*. Multiple notes in a column are dyads and chords.

```
                                                  w q q
            8       u       7       y             o p p
    i p z v b z p b n z p n d [ i d z p i p z p i u i i
```

What is this familiar melody?

```
nn//SS/ ..,,mmn //..,,m //..,,m nn//SS/ ..,,mmn   (S = space)
```

Are you able to play another song of your choice?

Challenge Modify the Karplus-Strong algorithm to synthesize a different instrument. Consider changing the excitation of the string (from white-noise to something more structured) or changing the averaging formula (from the average of the first two samples to a more complicated rule) or anything else you might imagine.

Having trouble with Guitar Heroine? Check to see if it is one of the following:

When I run `GuitarHeroineLite` for the first time, I hear no sound. What am I doing wrong? Make sure you have tested with the `main()` provided for `GuitarString`. If that works, it is likely something wrong with `pluck()` since the `main()` provided for `GuitarString` does not test that method. To diagnose the problem, print out the values of `sample()` and check that they become nonzero after you type *lower case* characters `'a'` and `'c'`.

When I run `GuitarHeroineLite`, I hear static (either just one click, and then silence or continual static). What am I doing wrong? It's likely that `pluck()` is working, but `tic()` is not. The best test is to run the `main()` provided for `GuitarString`.

How do I use `keyboard.indexOf(key)`? If `keyboard` is a `String` and `key` is a character, then `keyboard.indexOf(key)` return the integer index of the first occurrence of the character `key` in the string `keyboard` (or -1 if it does not occur).