```
    1: all: GuitarHero GStest
    2:
    3: GuitarHero: GuitarString.o GuitarHero.o RingBuffer.o
    4:         g++ -o GuitarHero GuitarString.o GuitarHero.o RingBuffer.o -lsfml-graphic
s -lsfml-window -lsfml-system -lsfml-audio -lboost_unit_test_framework
    5:
    6: GStest: GuitarString.o RingBuffer.o GStest.o
    7:         g++ -o GStest GuitarString.o GStest.o RingBuffer.o -lsfml-graphics -lsfml
-window -lsfml-system -lsfml-audio -lboost_unit_test_framework
    8:
    9: GStest.o: GStest.cpp GuitarString.hpp
   10:         g++ -c -g -Wall -ansi -pedantic GStest.cpp
   11:
   12: GuitarHero.o: GuitarHero.cpp GuitarString.hpp
   13:         g++ -c -g -Wall -ansi -pedantic GuitarHero.cpp
   14:
   15: GuitarString.o: GuitarString.cpp GuitarString.hpp
   16:         g++ -c -g -Wall -ansi -pedantic GuitarString.cpp
   17:
   18: RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
   19:         g++ -c -g -Wall -ansi -pedantic RingBuffer.cpp
   20:
   21: clean:
   22:         rm *.o GuitarHero GStest
```

```
    1: /*
    2:  Copyright 2015 Fred Martin, fredm@cs.uml.edu
    3:  Mon Mar 30 08:58:49 2015
    4:
    5:  based on Princeton's GuitarHeroLite.java
    6:  www.cs.princeton.edu/courses/archive/fall13/cos126/assignments/guitar.html
    7:
    8:  build with
    9:  g++ -Wall -c GuitarHeroLite.cpp -lsfml-system \
   10:  -lsfml-audio -lsfml-graphics -lsfml-window
   11:  g++ -Wall GuitarHeroLite.o RingBuffer.o GuitarString.o \
   12:  -o GuitarHeroLite -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window
   13:  */
   14:
   15: #include <SFML/Graphics.hpp>
   16: #include <SFML/System.hpp>
   17: #include <SFML/Audio.hpp>
   18: #include <SFML/Window.hpp>
   19:
   20: #include <math.h>
   21: #include <limits.h>
   22:
   23: #include <iostream>
   24: #include <string>
   25: #include <exception>
   26: #include <stdexcept>
   27: #include <vector>
   28:
   29: #include "RingBuffer.hpp"
   30: #include "GuitarString.hpp"
   31:
   32: #define CONCERT_A 440.0
   33: #define SAMPLES_PER_SEC 44100
   34: using namespace std;
   35:
   36: vector<sf::Int16> makeSamplesFromString(GuitarString& gs) {
   37:     std::vector<sf::Int16> samples;
   38:
   39:     gs.pluck();
   40:     int duration = 8;  // seconds
   41:     int i;
   42:     for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
   43:         gs.tic();
   44:         samples.push_back(gs.sample());
   45:     }
   46:
   47:     return samples;
   48: }
   49:
   50: int main() {
   51:     sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Guitar Hero Lite");
   52:     sf::Event event;
   53:     double freq;
   54:     unsigned key_index;
   55:     string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
   56:     vector < std::vector<sf::Int16> >samples(keyboard.length());
   57:     vector<sf::SoundBuffer> buffer(keyboard.length());
   58:     vector<sf::Sound> sound(keyboard.length());
   59:
   60:     for (int i = 0; i < keyboard.length(); i++) {
   61:         freq = CONCERT_A * pow(2, (i-24) / 12.0);
   62:         GuitarString gs = GuitarString(freq);
   63:         samples[i] = makeSamplesFromString(gs);
   64:         if(!buffer[i].loadFromSamples(&samples[i][0], samples[i].size(), 2, SAMPL
ES_PER_SEC))
```

```
65:             throw runtime_error("sf::SoundBuffer: failed to load from samples.");
66:         sound[i].setBuffer(buffer[i]);
67:     }
68:
69:     // we're reusing the freq and samples vars, but
70:     // there are separate copies of GuitarString, SoundBuffer, and Sound
71:     //   for each note
72:     //
73:     // GuitarString is based on freq
74:     // samples are generated from GuitarString
75:     // SoundBuffer is loaded from samples
76:     // Sound is set to SoundBuffer
77:
78:     while (window.isOpen()) {
79:         while (window.pollEvent(event)) {
80:             switch (event.type) {
81:                 case sf::Event::Closed:
82:                     window.close();
83:                     break;
84:
85:                 case sf::Event::TextEntered:
86:                     key_index = keyboard.find(event.text.unicode);
87:                     sound[key_index].play();
88:                     break;
89:
90:                 default:
91:                     break;
92:             }
93:
94:             window.clear();
95:             window.display();
96:         }
97:     }
98:     return 0;
99: }
```

```
 1: /*
 2:  Copyright 2015 Fred Martin, fredm@cs.uml.edu
 3:  Wed Apr  1 09:43:12 2015
 4:  test file for GuitarString class
 5:
 6:  compile with
 7:  g++ -c GStest.cpp -lboost_unit_test_framework
 8:  g++ GStest.o GuitarString.o RingBuffer.o -o GStest -lboost_unit_test_framework
 9:  */
10:
11: #define BOOST_TEST_DYN_LINK
12: #define BOOST_TEST_MODULE Main
13: #include <boost/test/unit_test.hpp>
14:
15: #include <vector>
16: #include <exception>
17: #include <stdexcept>
18:
19: #include "GuitarString.hpp"
20:
21: BOOST_AUTO_TEST_CASE(GS) {
22:     std::vector<sf::Int16> v;
23:
24:     v.push_back(0);
25:     v.push_back(2000);
26:     v.push_back(4000);
27:     v.push_back(-10000);
28:
29:     BOOST_REQUIRE_NO_THROW(GuitarString gs = GuitarString(v));
30:
31:     GuitarString gs = GuitarString(v);
32:
33:     // GS is 0 2000 4000 -10000
34:     BOOST_REQUIRE(gs.sample() == 0);
35:
36:     gs.tic();
37:     // it's now 2000 4000 -10000 996
38:     BOOST_REQUIRE(gs.sample() == 2000);
39:
40:     gs.tic();
41:     // it's now 4000 -10000 996 2988
42:     BOOST_REQUIRE(gs.sample() == 4000);
43:
44:     gs.tic();
45:     // it's now -10000 996 2988 -2988
46:     BOOST_REQUIRE(gs.sample() == -10000);
47:
48:     gs.tic();
49:     // it's now 996 2988 -2988 -4483
50:     BOOST_REQUIRE(gs.sample() == 996);
51:
52:     gs.tic();
53:     // it's now 2988 -2988 -4483 1984
54:     BOOST_REQUIRE(gs.sample() == 2988);
55:
56:     gs.tic();
57:     // it's now -2988 -4483 1984 0
58:     BOOST_REQUIRE(gs.sample() == -2988);
59:
60:     // a few more times
61:     gs.tic();
62:     BOOST_REQUIRE(gs.sample() == -4483);
63:     gs.tic();
64:     BOOST_REQUIRE(gs.sample() == 1984);
65:     gs.tic();
```

```
   66:     BOOST_REQUIRE(gs.sample() == 0);
   67: }
```

```
    1: //
    2: //  GuitarString.cpp
    3: //  ps5b
    4: //
    5: //  Created by Jingxian Shi on 3/26/18.
    6: //  Copyright Â© 2018 Jingxian Shi. All rights reserved.
    7: //
    8: #include <stdint.h>
    9: #include <cmath>
   10: #include <vector>
   11: #include "GuitarString.hpp"
   12:
   13: GuitarString::GuitarString(double frequency) {
   14:     int size = ceil(44100.0/frequency);
   15:     _rb = new RingBuffer(size);
   16:     _time = 0;
   17:     pluck();
   18: }
   19:
   20: GuitarString::GuitarString(std::vector<sf::Int16> init) {
   21:     _rb = new RingBuffer(static_cast<int>(init.size()));
   22:     for (int i = 0; i < init.size(); i++) {
   23:         _rb->enqueue(init[i]);
   24:     }
   25:     _time = 0;
   26: }
   27:
   28: GuitarString::~GuitarString() {
   29:     delete _rb;
   30: }
   31:
   32: void GuitarString::pluck() {
   33:     _rb->empty();
   34:     while (!_rb->isFull()) {
   35:         _rb->enqueue((sf::Int16)rand() & 0xffff);
   36:     }
   37: }
   38:
   39: void GuitarString::tic() {
   40:     sf::Int16 sample1 = _rb->dequeue();
   41:     sf::Int16 sample2 = _rb->peek();
   42:     double newUpdate = 0.996 * ((sample1 + sample2)/2.0);
   43:     _rb->enqueue(newUpdate);
   44:     _time++;
   45: }
   46:
   47: double GuitarString::sample() {
   48:     return _rb->peek();
   49: }
   50:
   51: int GuitarString::time() {
   52:     return _time;
   53: }
```

```
 1: //
 2: //  GuitarString.hpp
 3: //  ps5b
 4: //
 5: //  Created by Jingxian Shi on 3/26/18.
 6: //  Copyright Â© 2018 Jingxian Shi. All rights reserved.
 7: //
 8:
 9: #ifndef GuitarString_hpp
10: #define GuitarString_hpp
11:
12: #include <SFML/System.hpp>
13: #include <vector>
14: #include "RingBuffer.hpp"
15:
16: class GuitarString {
17:  public:
18:     explicit GuitarString(double frequency);
19:     explicit GuitarString(std::vector<sf::Int16> init);
20:     ~GuitarString();
21:     void pluck();
22:     void tic();
23:     double sample();
24:     int time();
25:  private:
26:     RingBuffer* _rb;
27:     int _time;
28: };
29:
30: #endif /* GuitarString_hpp */
```

```
 1: //
 2: //  RingBuffer.cpp
 3: //  ps5a
 4: //
 5: //  Created by Jingxian Shi on 3/19/18.
 6: //  Copyright Â© 2018 Jingxian Shi. All rights reserved.
 7: //
 8:
 9: #include "RingBuffer.hpp"
10: #include <exception>
11: #include <stdexcept>
12: #include <vector>
13: #include <string>
14:
15: RingBuffer::RingBuffer(int capacity) {
16:     if ( capacity < 1 )
17:         throw std::invalid_argument("Capacity must be greater than zero");
18:     _queue.resize(capacity);
19:     _capacity = capacity;
20:     _size = 0;
21:     _front = 0;
22:     _end = 0;
23: }
24:
25: void RingBuffer::enqueue(int16_t x) {
26:     if ( isFull() ) {
27:         throw std::runtime_error("Can’t enqueue to a full ring");
28:     }
29:     _queue[_end] = x;
30:     if (_end == (_capacity-1)) {
31:         _end = 0;
32:         _size++;
33:         return;
34:     }
35:     _end++;
36:     _size++;
37:     return;
38: }
39:
40: int16_t RingBuffer::dequeue() {
41:     if ( isEmpty() ) {
42:         throw std::runtime_error("Can’t dequeue to an empty RingBuffer");
43:     }
44:     int16_t item = _queue[_front];
45:     if ( _front == (_capacity-1) ) {
46:         _front = 0;
47:         _size--;
48:         return item;
49:     }
50:     _size--;
51:     _front++;
52:     return item;
53: }
54:
55: int16_t RingBuffer::peek() {
56:     if ( isEmpty() ) {
57:         throw std::runtime_error("Can’t peek an empty RingBuffer");
58:     }
59:     return _queue[_front];
60: }
61:
62: void RingBuffer::empty() {
63:     _front = 0;
64:     _end = 0;
65:     _size = 0;
```

```
66: }
```

```
 1: //
 2: //  RingBuffer.hpp
 3: //  ps5a
 4: //
 5: //  Created by Jingxian Shi on 3/19/18.
 6: //  Copyright Â© 2018 Jingxian Shi. All rights reserved.
 7: //
 8:
 9: #ifndef RingBuffer_hpp
10: #define RingBuffer_hpp
11:
12: #include <stdio.h>
13: #include <stdint.h>
14: #include <vector>
15:
16: class RingBuffer {
17:  public:
18:     explicit RingBuffer(int capacity);
19:     int size() {return _size;}
20:     bool isEmpty() {return (_size == 0);}
21:     bool isFull() {return (_size == _capacity); }
22:     void enqueue(int16_t x);
23:     int16_t dequeue();
24:     int16_t peek();
25:     void empty();
26:  private:
27:     std::vector<int16_t> _queue;
28:     int _size;
29:     int _capacity;
30:     int _front;
31:     int _end;
32: };
33:
34:
35: #endif /* RingBuffer_hpp */
36:
```