```
all: GuitarHero

GuitarHero: GuitarHero.o GuitarString.o RingBuffer.o
	g++ GuitarHero.o RingBuffer.o GuitarString.o -o GuitarHero -lboost_unit_test_framework -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window -g

GuitarHero.o: GuitarHero.cpp GuitarString.hpp RingBuffer.hpp
	g++ -c GuitarHero.cpp -Wall -Werror -ansi -pedantic -lboost_unit_test_framework -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window -g

RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
	g++ -c RingBuffer.cpp -Wall -Werror -ansi -pedantic -lboost_unit_test_framework -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window -g

GuitarString.o: GuitarString.cpp GuitarString.hpp
	g++ -c GuitarString.cpp -Wall -Werror -ansi -pedantic -lboost_unit_test_framework -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window -g

clean:
	 rm *.o GuitarHero *~ a.out
```

```
// Copyright 2015 <Angel Calcano>
// PS5b
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>
#include <math.h>
#include <limits.h>
#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include <vector>
#include "RingBuffer.hpp"
#include "GuitarString.hpp"
#define SAMPLES_PER_SEC 48400

std::vector< sf::Int16 > makeSamplesFromString(GuitarString *gs) {
  std::vector< sf::Int16 > samples;
  gs->pluck();
  int duration = 8;
  int i;
  for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
    gs->tic();
    samples.push_back(gs->sample());
  }
  return samples;
}
int main(int argc, char *argv[]) {
  sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Guitar Hero");
  sf::Event event;
  double freq;
  int i;
  std::vector< std::vector< int16_t >  > ado_smpl_strm;
  std::vector< sf::SoundBuffer > ado_smpl;
  std::vector< sf::Sound > SndBffer;
  std::vector< sf::Int16 > samples;
  std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/’ ";
  ado_smpl_strm.resize(37);//ALWAYS USE resize
  ado_smpl.resize(37);
  SndBffer.resize(37);
  for (i = 0; i < 37; i++) {
    freq = 220*pow(2, (i-24)/12.0);//changed from 440 to 220
    //GuitarString gs1(freq); making shallow copy
    GuitarString *gs1 = new GuitarString(freq);
    samples = makeSamplesFromString(gs1);
    ado_smpl[i].loadFromSamples( &samples[0], samples.size(), 2, SAMPLES_PER_SEC); // y
    SndBffer[i].setBuffer(ado_smpl[i]);
    //sf::SoundBuffer buf1;
    //sf::Sound sound1;
    //if (!buf1.loadFromSamples(&ado_smpl_strm[i][0], ado_smpl_strm[i].size(), 2, SAMPLES_PER_SEC))
    //throw std::runtime_error("sf::SoundBuffer: failed to load from samples.");
    //sound1.setBuffer(buf1);
    //ado_smpl.push_back(buf1);
    //sound1.setBuffer(ado_smpl[i]);
    //SndBffer.push_back(sound1);
  }
  int index;
  while (window.isOpen()) {
    while (window.pollEvent(event)) {
      switch (event.type) {
      case sf::Event::Closed:
        window.close();
        break;
      case sf::Event::TextEntered:
        index = keyboard.find(event.text.unicode);
        if ((unsigned)index != std::string::npos){
           SndBffer[index].play();
        }
        break;
      default:
        break;
      }
    }
    window.clear();
    window.display();
  }
  return 0;
}
```

```
// Copyright 2015 <Angel Zheondre Calcano>
// PS5b
#ifndef _GuitarString_
#define _GuitarString_
#include <math.h>
#include <limits.h>
#include <SFML/System.hpp>
#include <stdint.h>
#include <cstdlib>
#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include <vector>
#include "RingBuffer.hpp"

class GuitarString{
  RingBuffer *_j; int _size, _ticCount;

 public:
  explicit GuitarString(double freq) {
    if (freq < 1)
      throw std::runtime_error("Constructor frequency must be > than 0");
    _size = ceil((48400/freq)); //48400
    _j = new RingBuffer(_size);
    _ticCount = 0;
     for (int i = 0 ; i < _size; i++)
       _j->enqueue(0);
  }
  explicit GuitarString(std::vector< sf::Int16 > j) {
    _size = j.size();
    if (_size < 1)
      throw std::runtime_error("Empty Vector, Size must be > than 0 ");
    _j = new RingBuffer(_size);
    _ticCount = 0;
    for (int i = 0; i < _size; i++) {
      _j->enqueue((int16_t)j[i]);
    }
  }
  ~GuitarString() {
    delete _j; // delete *_j made it fail the test.
  }
  void pluck();
  int time();
  void tic();
  sf::Int16 sample();
};
#endif
```

```
// Copyright 2015 < Angel Zheondre Calcano>
// PS5b
#include <math.h>
#include <stdint.h>
#include <iostream>
#include <string>
#include <cstdlib>
#include <exception>
#include <stdexcept>
#include <vector>
#include "GuitarString.hpp"

void GuitarString::pluck() {
  if (_j->isEmpty())
    throw
      std::runtime_error("Can’t pluck, empty buffer.");
  int i; int16_t ran;
  for (i = 0 ; i < _size; i++)
    _j->dequeue(); // all 0s
  for (i = 0 ; i < _size; i++) {
    ran = (int16_t)(rand() & 0xffff);
    _j->enqueue(ran);
  }
  //std::cout << ran << " pluck" <<std::endl; //w
}
void GuitarString::tic() {
  if (_j->isEmpty())
    throw
      std::runtime_error("Can’t tic, empty buffer.");
  double num1, num2, result;
  int i;
  num1 = _j->dequeue();
  num2 = _j->peek();
  result = .996*.5*(num1 + num2);
  //std::cout<< result << std::endl;
  for (i = 0 ; i < _size - 1; i++)// this function seems weird.
       _j->enqueue(_j->dequeue());
  _j->enqueue(result);
  //std::cout<< _j->peek();
  _ticCount++;
}
int16_t GuitarString::sample() {
if (_j->isEmpty())
    throw
      std::runtime_error("Can’t peek, empty buffer.");
return _j->peek();
}
int GuitarString::time() {
  return _ticCount;
}
```

```
// Copyright 2015 <Angel Z’heondre Calcano>
// PS5b
#ifndef _RingBuffer_
#define _RingBuffer_
#include <stdint.h>
#include <iostream>
#include <string>
#include <vector>
#include <stdexcept>

class RingBuffer{
  int _size, _first, _last, _currentcapacity;
  std::vector< double > _buffer;
 public:
  explicit RingBuffer(int x): _size(x) {
    if (x < 1)
      throw std::runtime_error("Constructor capacity must be > than 0");
    _first = _last = _currentcapacity = 0;
    for (int i = 0 ; i < x; i++)
      _buffer.push_back(100);
   }
  void RB();
  int size();
  bool isEmpty();
  bool isFull();
  void enqueue(int16_t x);
  int16_t dequeue();
  int16_t peek();
};
#endif
```

```cpp
// Copyright 2015 <Angel Z’heondre Calcano>
// PS5b
#include <stdint.h>
#include <cstdlib>
#include <stdint.h>
#include <stdexcept>
#include <iostream>
#include <string>
#include <vector>
#include <SFML/System.hpp>
#include "RingBuffer.hpp"

int RingBuffer::size() { return _currentcapacity; }

bool RingBuffer::isEmpty() { return _buffer.empty(); }

bool RingBuffer::isFull() {
  if (_buffer.size() == (unsigned)_size)
    return true;
  if (_buffer.size() >(unsigned)_size)
    return false;
  else
    return false;
}

void RingBuffer::enqueue(int16_t x) {
  if (_currentcapacity == _size) {
    throw
      std::runtime_error("Can’t enqueue on a full ring");
  }
  if( _currentcapacity > _size) {
    throw
      std::runtime_error("Can’t enqueue on a full ring");
  }
  _buffer[_last] = x;
  _last++;
  _currentcapacity++;
  if (_last == _size)
    _last = 0;
}

int16_t RingBuffer::dequeue() {
  if (_currentcapacity <= 0)
    throw
      std::runtime_error(" Can’t dequeue from empty ring");
  double _hold;
  _hold = _buffer[_first];
  _first++;
  _currentcapacity--;
  if (_first == _last) _first = _last = 0;
  if (_first == _size) _first = 0;
  return _hold;
}

int16_t RingBuffer::peek() {
  if (_currentcapacity == 0)
    throw
      std::runtime_error("Can’t peek on an empty ring");
  if (_buffer.empty())
    throw
      std::runtime_error("Can’t peek on an empty vector array");
  return _buffer[_first];
}
```

```
/*
  Copyright 2015 Fred Martin, fredm@cs.uml.edu
  Wed Apr  1 09:43:12 2015
  test file for GuitarString class

  compile with
  g++ -c GStest.cpp -lboost_unit_test_framework
  g++ GStest.o GuitarString.o RingBuffer.o -o GStest -lboost_unit_test_framework
*/

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

#include <vector>
#include <exception>
#include <stdexcept>

#include "GuitarString.hpp"

using namespace std;

BOOST_AUTO_TEST_CASE(GS) {
  vector<sf::Int16> v;

  v.push_back(0);
  v.push_back(2000);
  v.push_back(4000);
  v.push_back(-10000);

  //BOOST_REQUIRE_NO_THROW(GuitarString gs = GuitarString(v));
  BOOST_REQUIRE_NO_THROW(GuitarString gs(v));
  GuitarString gs = GuitarString(v);

  // GS is 0 2000 4000 -10000
  BOOST_REQUIRE(gs.sample() == 0);

  gs.tic();
  // it’s now 2000 4000 -10000 996
  BOOST_REQUIRE(gs.sample() == 2000);

  gs.tic();
  // it’s now 4000 -10000 996 2988
  BOOST_REQUIRE(gs.sample() == 4000);

  gs.tic();
  // it’s now -10000 996 2988 -2988
  BOOST_REQUIRE(gs.sample() == -10000);

  gs.tic();
  // it’s now 996 2988 -2988 -4483
  BOOST_REQUIRE(gs.sample() == 996);

  gs.tic();
  // it’s now 2988 -2988 -4483 1984
  BOOST_REQUIRE(gs.sample() == 2988);

  gs.tic();
  // it’s now -2988 -4483 1984 0
  BOOST_REQUIRE(gs.sample() == -2988);

  // a few more times
  gs.tic();
  BOOST_REQUIRE(gs.sample() == -4483);
  gs.tic();
  BOOST_REQUIRE(gs.sample() == 1984);
  gs.tic();
  BOOST_REQUIRE(gs.sample() == 0);
}
```