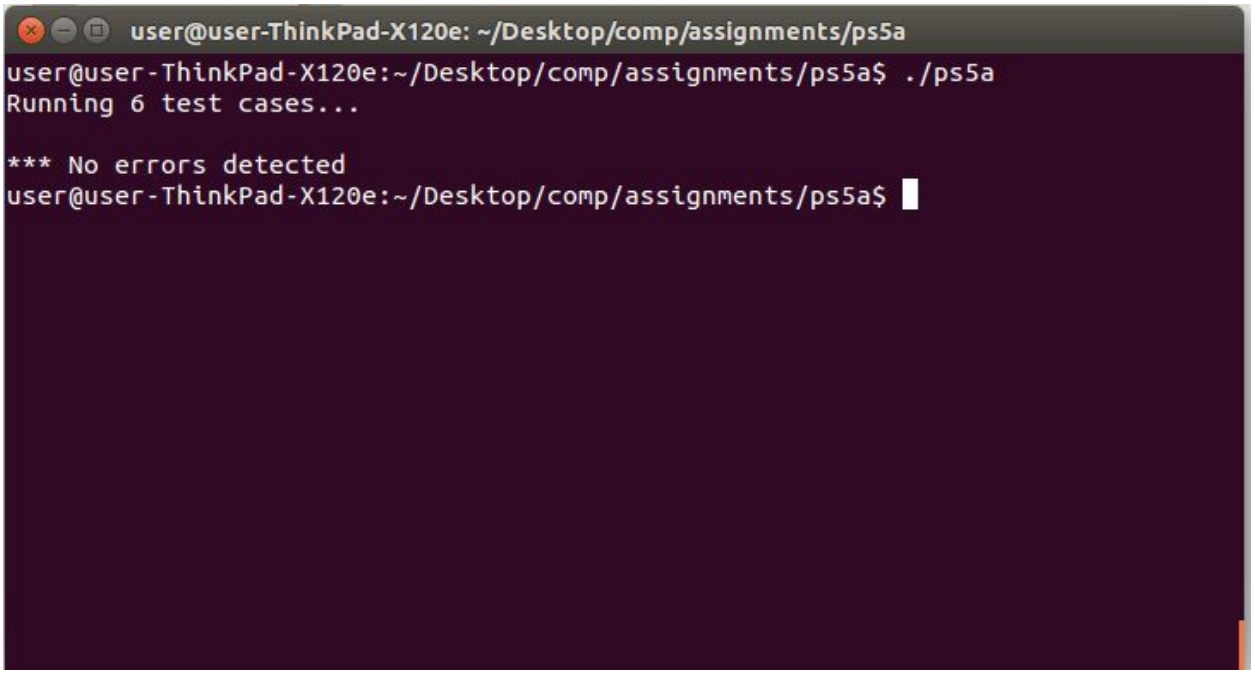
PS5 RingBuffer and GuitarHero:

This assignment had us go to the following link:
http://www.cs.princeton.edu/courses/archive/spr15/cos126/assignments/guitar.html

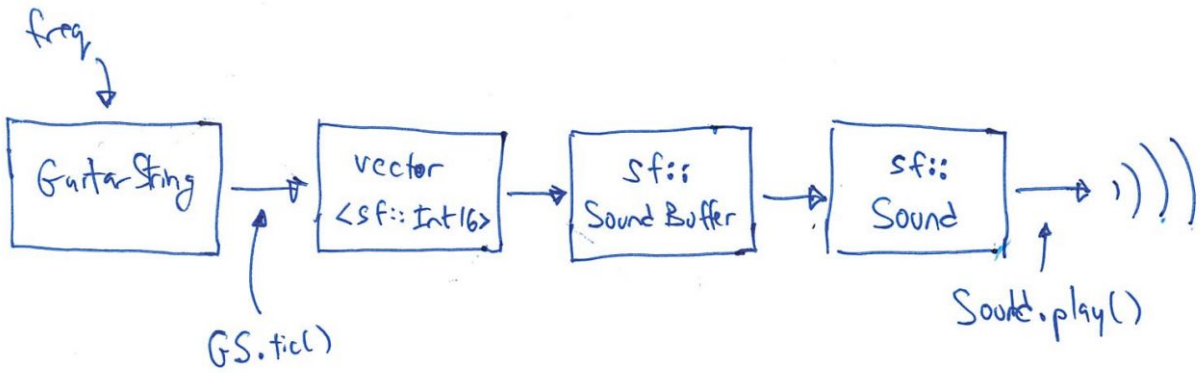
Similar to each of the two part assignments, part A covers the basics of what needs to be used as a template for part B. The first part deals with debugging and testing that utilizes the Boost functions BOOST_REQUIRE_THROW and BOOST_REQUIRE_NO_THROW to verify that the code properly throws the specified expectations when it is required. But it must execute ALL of the methods from the RingBuffer; if the vector is empty, fill, enqueuing, dequeuing, and peeking. A testing file was given to us though, so there was no need to make any random test cases.

Looks like this once you execute the program:

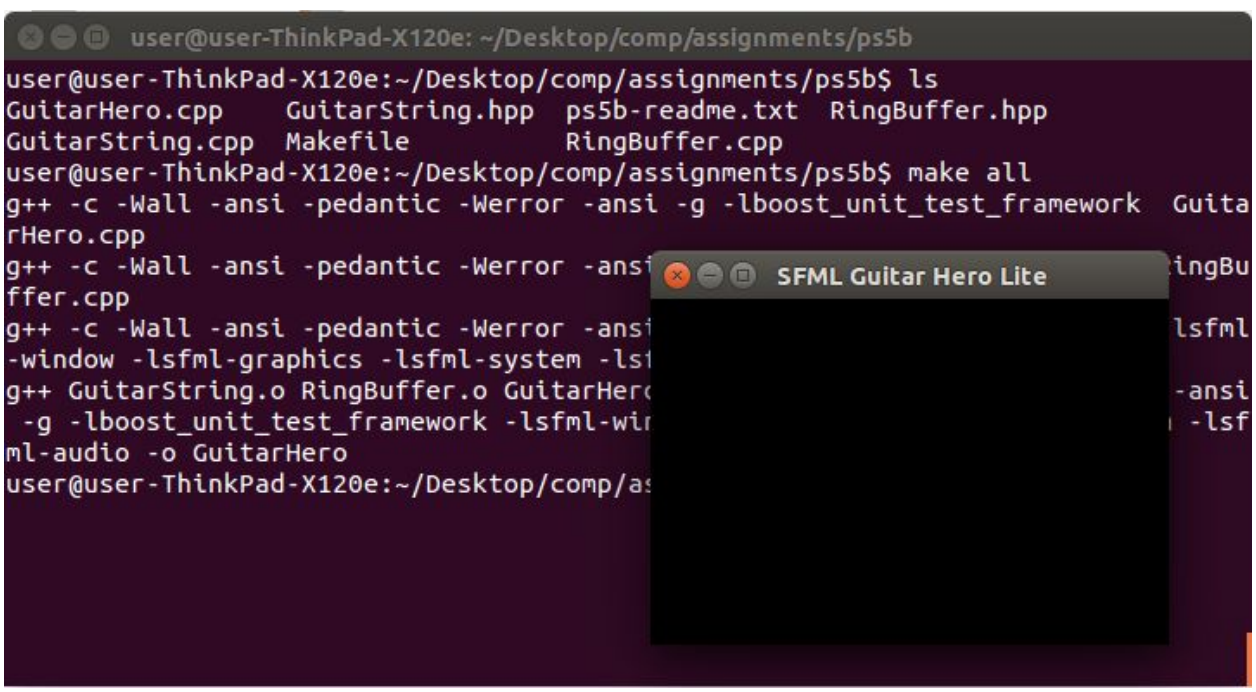


Part B of the assignment required us to implement the Karplus-Strong guitar string simulation, and generate a stream of string samples for audio playback under keyboard control.. We're given the GuitarString API to work with. *Note I did not finish the assignment correctly when I first attempted the task. However, one of the tricky parts was getting the audio to generate and playing the audio objects when the event occurs. I was able to figure it out though(just now actually) and fix my code to finish it.

Using the SoundBuffer from sf::Int16s the sound object may be played like this::



We are able to accomplish because of the Karplus-Strong algorithm along with the ring buffer files from part A. The feedback mechanism determines the frequency of the sound, and the averaging operation which serves as a gentle low-pass filter.



I wasn't able to get a picture to display when the program is executed, however when the user press the key-events "A" and "C" you should be able to hear the audio object. Overall the assignment was quite simple as I didn't run into any major problems except syntax ones. That was a mistake on my part though as I wasn't able to figure it out until now.

```
CC = g++
OFLAGS = -c -Wall -ansi -pedantic -Werror -ansi -g -lboost_unit_test_framework
CFLAGS = -Wall -ansi -pedantic -Werror -ansi -g -lboost_unit_test_framework
LFLAGS = -lsfml-window -lsfml-graphics -lsfml-system -lsfml-audio

all: GuitarHero

GuitarHero: GuitarHero.o RingBuffer.o GuitarHero.o
        $(CC) GuitarString.o RingBuffer.o GuitarHero.o $(CFLAGS) $(LFLAGS) -o GuitarHero

GuitarHero.o: GuitarHero.cpp
        $(CC) $(OFLAGS) $(LFLAGSs)  GuitarHero.cpp

RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
        $(CC) $(OFLAGS) RingBuffer.cpp

GuitarString.o: GuitarString.cpp GuitarString.hpp
        $(CC) $(OFLAGS) GuitarString.cpp

clean:
        \rm -f *.o *~ RingBuffer test ps5b GuitarString GuitarHero
```

```
 0: /*
 1:   Copyright 2015 Fred Martin, fredm@cs.uml.edu
 2:   Mon Mar 30 08:58:49 2015
 3:
 4:   based on Princeton’s GuitarHeroLite.java
 5:   www.cs.princeton.edu/courses/archive/fall13/cos126/assignments/guitar.html
 6:
 7:   build with
 8:   g++ -Wall -c GuitarHeroLite.cpp -lsfml-system \
 9:      -lsfml-audio -lsfml-graphics -lsfml-window
10:   g++ -Wall GuitarHeroLite.o RingBuffer.o GuitarString.o \
11:    -o GuitarHeroLite -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window
12: */
13:
14: #include <SFML/Graphics.hpp>
15: #include <SFML/System.hpp>
16: #include <SFML/Audio.hpp>
17: #include <SFML/Window.hpp>
18:
19: #include <math.h>
20: #include <limits.h>
21:
22: #include <iostream>
23: #include <string>
24: #include <exception>
25: #include <stdexcept>
26: #include <vector>
27:
28: #include "RingBuffer.hpp"
29: #include "GuitarString.hpp"
30:
31: #define CONCERT_A 220.0
32: #define SAMPLES_PER_SEC 44100
33:
34: vector<sf::Int16> makeSamplesFromString(GuitarString gs) {
35:   std::vector<sf::Int16> samples;
36:
37:   gs.pluck();
38:   int duration = 8;  // seconds
39:   int i;
40:   for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
41:     gs.tic();
42:     samples.push_back(gs.sample());
43:   }
44:
45:   return samples;
46: }
47:
48: int main() {
49:   sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Guitar Hero Lite");
50:   sf::Event event;
51:   double freq;
52:   vector<sf::Int16> samples;
53:
54:   // we’re reusing the freq and samples vars, but
55:   // there are separate copies of GuitarString, SoundBuffer, and Sound
56:   //   for each note
57:   //
58:   // GuitarString is based on freq
59:   // samples are generated from GuitarString
60:   // SoundBuffer is loaded from samples
61:   // Sound is set to SoundBuffer
62:
```

```
  freq = CONCERT_A;
  GuitarString gs1 = GuitarString(freq);
  sf::Sound sound1;
  sf::SoundBuffer buf1;
  samples = makeSamplesFromString(gs1);
  if (!buf1.loadFromSamples(&samples[0], samples.size(), 2, SAMPLES_PER_SEC))
    throw std::runtime_error("sf::SoundBuffer: failed to load from samples.");
  sound1.setBuffer(buf1);

  freq = CONCERT_A * pow(2, 3.0/12.0);
  GuitarString gs2 = GuitarString(freq);
  sf::Sound sound2;
  sf::SoundBuffer buf2;
  samples = makeSamplesFromString(gs2);
  if (!buf2.loadFromSamples(&samples[0], samples.size(), 2, SAMPLES_PER_SEC))
    throw std::runtime_error("sf::SoundBuffer: failed to load from samples.");
  sound2.setBuffer(buf2);

  while (window.isOpen()) {
    while (window.pollEvent(event)) {
      switch (event.type) {
      case sf::Event::Closed:
        window.close();
        break;

      case sf::Event::KeyPressed:
        switch (event.key.code) {
        case sf::Keyboard::A:
          sound1.play();
          break;
        case sf::Keyboard::C:
          sound2.play();
          break;
        default:
          break;
        }

      default:
        break;
      }

      window.clear();
      window.display();
    }
  }
  return 0;
}
```

```
#include "GuitarString.hpp"
#include <stdint.h>
#include <math.h>
#include <SFML/Audio.hpp>
#include <vector>
#include <iostream>
#include "RingBuffer.hpp"

GuitarString::GuitarString(double frequency)
{
        count = 0;
        N = ceil(frequency);
        ptrRB = new RingBuffer(N);
        while ((*ptrRB).isEmpty())
        (*ptrRB).enqueue(0);
}

GuitarString::GuitarString(std::vector<sf::Int16> init)
{
        count = 0;
        N = init.size();
        ptrRB = new RingBuffer(N);
        for (std::vector<sf::Int16>::
        iterator it = init.begin(); it != init.end(); ++it)
        (*ptrRB).enqueue(*it);
}

GuitarString:: Ì\203GuitarString()
{
        delete ptrRB;
}

void GuitarString::pluck()
{
        while (!(*ptrRB).isEmpty())
        (*ptrRB).dequeue();
                while (!(*ptrRB).isFull())
                (*ptrRB).enqueue((sf::Int16)(rand() & 0xffff));
}

void GuitarString::tic()
{
        int16_t front = (*ptrRB).dequeue();
        int16_t frontNext = (*ptrRB).peek();
        float result = ((front + frontNext)/2) * 0.996;
        (*ptrRB).enqueue(result);
}

sf::Int16 GuitarString::sample()
{
        return (*ptrRB).peek();
}

int GuitarString::time()
{
        return count++;
}
```

```
#ifndef GS_HPP_
#define GS_HPP_

#include <vector>
#include <SFML/Audio.hpp>
#include <stdint.h>
#include "RingBuffer.hpp"
class GuitarString
{
        private:
        RingBuffer *ptrRB;
        int N;

        public:
        explicit GuitarString(double frequency);
        explicit GuitarString(std::vector<sf::Int16> init);
        ~GuitarString();
        void pluck();
        void tic();
        sf::Int16 sample();
        int time();
        int count;
};
#endif
```

```
#include <stdint.h>
#include <iostream>
#include <vector>
#include <stdexcept>
#include "RingBuffer.hpp"

//Default constructor
RingBuffer::RingBuffer(int capacity)
{
        //Checking to see if the capacity is less than 1
        if(capacity <= 0)
        {
                throw invalid_argument("Error*: capacity must be greater than zero.");
        }
        ringbuffer.reserve(capacity);
}

//Returns the size of the vector
int RingBuffer::size()
{
        return ringbuffer.size();
}

//Function to check to see if the vector is empty
bool RingBuffer::isEmpty()
{
        //Checking to see if the size is equal to zero, if it passes then it will retur
n true
        if(size() == 0)
                return 1;
        else
                return 0;

}
//Function to check to see if the vector is full
bool RingBuffer::isFull()
{
        int current; //creating an integer for the current capacity of the vector
        current = ringbuffer.capacity(); //assigning value of the current capacity
        //Checking to see if the size is equal to the current capacity
        if(size() == current)
                return 1;
        else
                return 0;
}

//Function which allows us to enqueue a type "double" onto the vector
void RingBuffer::enqueue(int16_t x)
{
        //Checking to see if the vector is occupied
        if(!isFull())
                //if so then we use the push_back function to push the type onto the ve
ctor
                ringbuffer.push_back(x);
        //However if we have an error it displays an error message
        else
                throw runtime_error("Enqueue error*: Can’t enqueue to a full ring");
}

//Function which allows us to dequeue a type from the vector
int16_t RingBuffer::dequeue()
{
        //creating a temp variable
```

```
        int16_t temp;
        //setting equal to the front of the vector
        temp = ringbuffer.front();
        //checking to see if the vector is empty or not
        if(!isEmpty())
                ringbuffer.erase(ringbuffer.begin(), ringbuffer.begin() + 1);
        else
                throw runtime_error("Dequeue error*: unable to remove from an empty rin
g");
        return temp;
}
int16_t RingBuffer::peek()
{
        if(!isEmpty())
                return ringbuffer.front();
        else
                throw runtime_error("Peek error*: unable to peek at an empty ring");
}
```

```
#ifndef RB_HPP_
#define RB_HPP_

#include <stdint.h>
#include <iostream>
#include <vector>
#include <stdexcept>

using namespace std;

class RingBuffer
{
        private:
                //I decided to use a data struct of vectors for the enqueue/dequeue
                std::vector<int16_t> ringbuffer;

        public:
                //creates an empty ring buffer, with given max capacity
                explicit RingBuffer(int capacity);
                // returns number of items currently in the buffer
                int size();
                //is the buffer empty(size = zero?)
                bool isEmpty();
                //is the buffer full(size = capacity)
                bool isFull();
                //add item x to the end
                void enqueue(int16_t x);
                //delete and return item from the front
                int16_t dequeue();
                 //return(doesn't delete) item from the front
                int16_t peek();
};

#endif
```