

Ps4a Source Code: Makefile

```
1 CFLAGS = -Wall -Werror -std=c++11 -pedantic
2 DEPS = -lboost_unit_test_framework
3
4
5 ps4a: test.o RingBuffer.o
6     g++ test.cpp RingBuffer.cpp headers/RingBuffer.h -o ps4a $(CFLAGS) $(DEPS)
7
8 test.o:
9     g++ test.cpp RingBuffer.cpp headers/RingBuffer.h -o test.o $(CFLAGS) $(DEPS)
10
11 RingBuffer.o:
12     g++ -c RingBuffer.cpp headers/RingBuffer.h $(CFLAGS)
13
14 clean:
15     rm ps4a
16     rm *.o
17     rm headers/*.gch
18     rm *.out
```

Ps4b Source Code: Makefile

```
1 CC= g++
2 CFLAGS= -Wall -Werror -std=c++11
3 DEPS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
4
5 KSGuitarSim:    KSGuitarSim.o StringSound.o RingBuffer.o
6     $(CC) KSGuitarSim.o StringSound.o RingBuffer.o -o KSGuitarSim $(DEPS)
7
8 main.o: main.cpp StringSound.h
9     $(CC) -c main.cpp StringSound.h $(CFLAGS) $(DEPS)
10
11 StringSound.o: StringSound.cpp StringSound.h
12     $(CC) -c StringSound.cpp StringSound.h $(CFLAGS) $(DEPS)
13
14 RingBuffer.o: RingBuffer.cpp RingBuffer.h
15     $(CC) -c RingBuffer.cpp RingBuffer.h $(CFLAGS) $(DEPS)
16
17 clean:
18     rm *.o
19     rm *.gch
20     rm KSGuitarSim
```

Ps4a Source Code: test.cpp

```
1 // Copyright 2020 John Simonson
2
3 #define BOOST_TEST_DYN_LINK
4 #define BOOST_TEST_MODULE Main
5 #include <boost/test/unit_test.hpp>
6 #include "headers/RingBuffer.h"
7
8
9 // construtor tests
10 BOOST_AUTO_TEST_CASE(constructor) {
11     BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
12     BOOST_REQUIRE_THROW(RingBuffer(-1), std::invalid_argument);
13     BOOST_REQUIRE_NO_THROW(RingBuffer(21));
14 }
15
16 // size test
17 BOOST_AUTO_TEST_CASE(size) {
18     RingBuffer temp(21);
19     BOOST_REQUIRE(temp.size() == 0);
20 }
21
22 // isEmpty test
23 BOOST_AUTO_TEST_CASE(isEmpty) {
24     RingBuffer temp(21);
25     BOOST_REQUIRE(temp.isEmpty() == true);
26 }
27
28 // isFull test
29 BOOST_AUTO_TEST_CASE(isFull) {
30     RingBuffer temp(21);
31     BOOST_REQUIRE(temp.isFull() == false);
32 }
33
34 // enqueue test
35 BOOST_AUTO_TEST_CASE(enqueue) {
36     RingBuffer temp(1);
37     BOOST_REQUIRE(temp.size() == 0);
38     temp.enqueue(50);
39     BOOST_REQUIRE(temp.size() == 1);
40     BOOST_REQUIRE_THROW(temp.enqueue(21), std::runtime_error);
41 }
42
43 // dequeue test
```

```

44 BOOST_AUTO_TEST_CASE(dequeue) {
45     RingBuffer temp(21);
46     BOOST_REQUIRE_THROW(temp.dequeue(), std::runtime_error);
47 }
48
49
50 // peek test
51 BOOST_AUTO_TEST_CASE(peek) {
52     RingBuffer temp(21);
53     BOOST_REQUIRE_THROW(temp.peek(), std::runtime_error);
54 }

```

Ps4b Source Code: KSGuitarSim.cpp

```

1  #include <SFML/Graphics.hpp>
2  #include <SFML/System.hpp>
3  #include <SFML/Audio.hpp>
4  #include <SFML/Window.hpp>
5
6  #include <math.h>
7  #include <limits.h>
8
9  #include <iostream>
10 #include <string>
11 #include <exception>
12 #include <stdexcept>
13 #include <vector>
14
15 #include "RingBuffer.h"
16 #include "StringSound.h"
17
18 #define CONCERT_A 440.0
19 #define SAMPLES_PER_SEC 44100
20 const int num_of_keys = 37;
21 std::string keyboard = "q2we4r5ty7u8i9op-=[zxdcfvgbnjmk,.;/' ";
22 std::vector<sf::Int16> makeSamples(StringSound gs) {
23     std::vector<sf::Int16> samples;
24
25     gs.pluck();
26     int duration = 8; // seconds
27     int i;
28     for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
29         gs.tic();
30         samples.push_back(gs.sample());

```

```

31     }
32
33     return samples;
34 }
35
36 int main() {
37     sf::RenderWindow window(sf::VideoMode(300, 200), "ps4b");
38     sf::Event event;
39     double freq;
40     std::vector<sf::Int16> samples;
41     std::vector<std::vector<sf::Int16>> vector_of_samples(num_of_keys);
42     std::vector<sf::SoundBuffer> sound_buffer(num_of_keys);
43     std::vector<sf::Sound> sounds(num_of_keys);
44     for(int i = 0; i < num_of_keys; i++){
45         freq = CONCERT_A * pow(2, 3.0/12.0);
46         StringSound gs2 = StringSound(freq);
47         samples = makeSamples(gs2);
48         vector_of_samples[i] = samples;
49
50         sound_buffer[i].loadFromSamples(&vector_of_samples[i][0], vector_of_samples[i].
51
52         sounds[i].setBuffer(sound_buffer[i]);
53     }
54
55     while (window.isOpen()) {
56         while (window.pollEvent(event)) {
57             if (event.type == sf::Event::Closed)
58                 window.close();
59             if(event.type == sf::Event::TextEntered){
60                 char ascii = (char)(event.text.unicode);
61                 for(int i = 0; i < num_of_keys; i++){
62                     if(keyboard[i] == ascii)
63                         sounds[i].play();
64                 }
65                 window.clear();
66                 window.display();
67             }
68         }
69     }
70     return 0;
71 }

```

Ps4a Source Code: RingBuffer.hpp

```

1 // Copyright 2020 John Simonson
2 #ifndef RINGBUFFER_H_
3 #define RINGBUFFER_H_
4 #include<stdint.h>
5 #include<iostream>
6 #include<queue>
7
8 class RingBuffer{
9     public:
10 RingBuffer(int capacity);
11 // create an empty ring buffer, with given max capacity
12 const double ENERGY_DECAY_FACTOR = 0.996;
13 int     size(); // return number of items currently in the buffer
14 bool    isEmpty(); // is the buffer empty (size equals zero)?
15 bool    isFull(); // is the buffer full (size equals capacity)?
16 void    enqueue(int16_t x); // add item x to the end
17 int16_t dequeue(); // delete and return item from the front
18 int16_t peek(); // return (but do not delete) item from the front
19     private:
20 int front;
21 int back;
22 int Capacity;
23 int Size;
24 std::queue<int16_t> ringBuffer;
25 };
26 #endif

```

Ps4b Source Code: StringSound.hpp

```

1 #ifndef STRINGSOUND_H
2 #define STRINGSOUND_H
3 #include <SFML/Audio.hpp>
4 #include <string>
5 #include "RingBuffer.h"
6 #include <memory>
7 #include <stdint.h>
8 #include<vector>
9
10 class StringSound {
11     public:
12         StringSound(double frequency); // create a guitar string sound of the
13                                         // given frequency using a sampling rate
14                                         // of 44,100

```

```

15     StringSound(std::vector<sf::Int16> init);           // create a guitar string wi
16                                     // size and initial values are given by
17                                     // the vector
18     StringSound(RingBuffer init);
19     void pluck();                                       // pluck the guitar string by replacing
20                                     // the buffer with random values,
21                                     // representing white noise
22     void tic();                                         // advance the simulation one time step
23     sf::Int16 sample();                               // return the current sample
24     int time();                                        // return number of times tic was called
25                                     // so far
26     void generate();
27 private:
28     std::shared_ptr<RingBuffer> ring_buffer;
29     int tics;
30 };
31
32 #endif

```

Ps4a Source Code: RingBuffer.cpp

```

1 // Copyright 2020 John Simonson
2 #include "RingBuffer.h"
3 using std::queue;
4 using std::runtime_error;
5 using std::invalid_argument;
6
7
8 RingBuffer::RingBuffer(int capacity) {
9     if (capacity < 1) {
10         throw
11             invalid_argument("Capacity must be 1 or greater");
12     }
13     front = 0;
14     back = 0;
15     Capacity = capacity;
16     Size = 0;
17 }
18
19 int RingBuffer::size() {
20     return ringBuffer.size();
21 }
22
23 bool RingBuffer::isEmpty() {

```

```

24     return ringBuffer.empty();
25 }
26
27 bool RingBuffer::isFull() {
28     if (Size == Capacity) {
29         return true;
30     }
31     return false;
32 }
33
34 void RingBuffer::enqueue(int16_t x) {
35     if (Size == Capacity) {
36         throw
37         runtime_error("cannot push onto full buffer");
38     }
39     ringBuffer.push(x);
40     Size++;
41     return;
42 }
43
44 int16_t RingBuffer::dequeue() {
45     if (ringBuffer.empty()) {
46         throw
47         runtime_error("cannot dequeue an empty buffer");
48     }
49     int16_t temp = ringBuffer.front();
50     ringBuffer.pop();
51     Size--;
52     return temp;
53 }
54
55 int16_t RingBuffer::peek() {
56     if (ringBuffer.empty()) {
57         throw
58         runtime_error("cannot peek at an empty buffer");
59     }
60     return (ringBuffer.front());
61 }

```

Ps4b Source Code: StringSound.cpp

```

1  #include "StringSound.h"
2  #include <random>
3  #define Rand std::uniform_int_distribution;

```

```

4  #define Gen std::default_random_engine;
5
6  const int16_t MIN = 1 << 15;
7  const int16_t MAX = ~MIN;
8
9  StringSound::StringSound(double frequency){
10     if(frequency <= 0){
11         throw
12         std::invalid_argument("frequency must be greater than zero");
13     }
14     ring_buffer = std::shared_ptr<RingBuffer>(new RingBuffer(frequency));
15     tics = 0;
16 }
17
18 StringSound::StringSound(RingBuffer rb){
19     ring_buffer = std::shared_ptr<RingBuffer>(new RingBuffer(rb));
20     tics = 0;
21 }
22
23 StringSound::StringSound(std::vector<sf::Int16> init ){
24     ring_buffer = std::shared_ptr<RingBuffer>(new RingBuffer(init.size()));
25     tics = 0;
26 }
27
28 void StringSound::pluck(){
29     std::default_random_engine rand_gen;
30     std::uniform_int_distribution<int16_t> gen = std::uniform_int_distribution<i
31     while(!ring_buffer->isFull())
32         ring_buffer->enqueue(gen(rand_gen));
33 }
34
35 sf::Int16 StringSound::sample(){
36     return ring_buffer->peek();
37 }
38
39 void StringSound::tic(){
40     tics++;
41     int16_t temp1 = ring_buffer->dequeue();
42     int16_t temp2 = ring_buffer->peek();
43     temp1 = temp1 + temp2;
44     temp1 = temp1 / 2;
45     temp1 = temp1 * 0.996;
46     ring_buffer->enqueue(temp1);
47 }
48
49 int StringSound::time(){

```



```
50  return tics;  
51  
52  }
```
