

PS4 CircularBuffer and StringSound

What is the assignment?

For part A of the assignment, it is pretty much setting up the foundations of simulating a guitar string on a keyboard which is possible by first creating and implementing a CircularBuffer class that we will utilize in part b of this project.

And for part B of the assignment, it is utilizing our recently built CircularBuffer class inside our StringSound class that helps stores sound for us to utilize when we play a guitar on a keyboard.

And at the end of the day, the final product will be able to produce guitar sounds when pressing certain keyboard buttons.

What I learned from this assignment:

- 1) I learned more about the sf::Event variables and some of its functions to utilize for the assignment.
- 2) I also learned how to build a deque and how it works as well.
- 3) I learned how to make a circular buffer and utilize lambdas.

What I accomplished in this assignment:

- 1) I successfully managed to simulate a guitar on my laptop.
- 2) I also managed to implement lambda expressions and smart pointers within my StringSound class implementation file.
- 3) Additionally, I managed to not use multiple if statements for each keyboard button, reducing the lines of code needed and redundancy.
- 4) I successfully accomplished making my own lambda expressions.
- 5) Managed to boost test all my methods or class functions to see if they work as intended.

Any Issues I had in this assignment:

For part A of the assignment, I did not have any serious issues other than the fact that I struggled initially on making the circular buffer work.

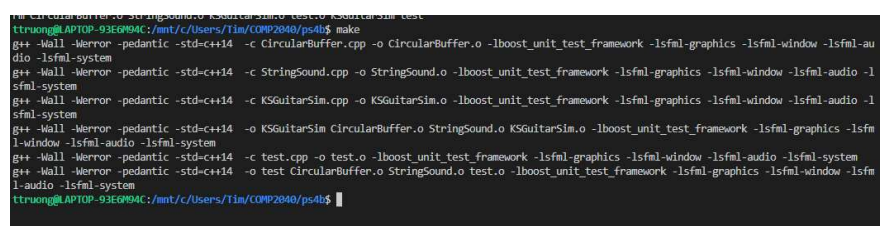
For part B of the assignment, I had a challenging time setting up pulseaudio which is necessary to test my main to see if the guitar plays as intended.

Key Algorithms, Data Structures, or OO Designs used for this assignment:

For part A, I utilized a vector to simulate how a Circular Buffer would normally work and I created a couple of private data members to help determine if the Circular Buffer is full or not.

For part B, I utilized dequeues to make my circular buffer implementation more efficient and vectors to store the sound buffers for me to utilize when playing the guitar. By doing this, I can be able to freely access all the 37 notes of a guitar when I press a keyboard button on my keyboard.

Screenshot(s):



```

ttruong@LAPTOP-93E69BAC:~/mnt/c/Users/tim/COMP2040/ps4b$ make
g++ -Wall -Werror -pedantic -std=c++14 -c CircularBuffer.cpp -o CircularBuffer.o -lboost_unit_test_framework -lsfml-graphics -lsfml-window -lsfml-audio -lsfml-system
g++ -Wall -Werror -pedantic -std=c++14 -c StringSound.cpp -o StringSound.o -lboost_unit_test_framework -lsfml-graphics -lsfml-window -lsfml-audio -lsfml-system
g++ -Wall -Werror -pedantic -std=c++14 -c KSGuitarSim.cpp -o KSGuitarSim.o -lboost_unit_test_framework -lsfml-graphics -lsfml-window -lsfml-audio -lsfml-system
g++ -Wall -Werror -pedantic -std=c++14 -o KSGuitarSim CircularBuffer.o StringSound.o KSGuitarSim.o -lboost_unit_test_framework -lsfml-graphics -lsfml-window -lsfml-audio -lsfml-system
g++ -Wall -Werror -pedantic -std=c++14 -c test.cpp -o test.o -lboost_unit_test_framework -lsfml-graphics -lsfml-window -lsfml-audio -lsfml-system
g++ -Wall -Werror -pedantic -std=c++14 -o test CircularBuffer.o StringSound.o test.o -lboost_unit_test_framework -lsfml-graphics -lsfml-window -lsfml-audio -lsfml-system
ttruong@LAPTOP-93E69BAC:~/mnt/c/Users/tim/COMP2040/ps4b$

```



```
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps4b$ ./test
Running 7 test cases...
```

```
*** No errors detected
```

```
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps4b$
```

```
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps4b$ ./KSGuitarSim
libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
Warning: The created OpenGL context does not fully meet the settings that were requested
Requested: version = 1.1 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Created: version = 0.0 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Setting vertical sync failed
```

```
stem
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps4a$ ./CircularBuffer
Running 5 test cases...

*** No errors detected
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps4a$
```

```
1: CC = g++
2: CFLAGS = -Wall -Werror -pedantic -std=c++14
3: LIBS = -lboost_unit_test_framework
4: OBJECTS = CircularBuffer.o StringSound.o KSGuitarSim.o
5: TOBJECTS = CircularBuffer.o StringSound.o test.o
6: SFML = -lsfml-graphics -lsfml-window -lsfml-audio -lsfml-system
7:
8: all: KSGuitarSim test
9: KSGuitarSim: $(OBJECTS)
10:      $(CC) $(CFLAGS) -o KSGuitarSim $(OBJECTS) $(LIBS) $(SFML)
11: test: $(TOBJECTS)
12:      $(CC) $(CFLAGS) -o test $(TOBJECTS) $(LIBS) $(SFML)
13: CircularBuffer.o: CircularBuffer.cpp CircularBuffer.h
14:      $(CC) $(CFLAGS) -c CircularBuffer.cpp -o CircularBuffer.o $(LIBS)
$(SFML)
15: StringSound.o: StringSound.cpp StringSound.h
16:      $(CC) $(CFLAGS) -c StringSound.cpp -o StringSound.o $(LIBS) $(SFML)
L)
17: KSGuitarSim.o: KSGuitarSim.cpp
18:      $(CC) $(CFLAGS) -c KSGuitarSim.cpp -o KSGuitarSim.o $(LIBS) $(SFML)
L)
19: test.o: test.cpp
20:      $(CC) $(CFLAGS) -c test.cpp -o test.o $(LIBS) $(SFML)
21: lint:
22:      cpplint *.cpp *.h
23: clean:
24:      rm $(OBJECTS) test.o KSGuitarSim test
```

```
1: // Copyright [2022] Tim Truong
2:
3: /*****
4:  *Name: Tim Truong
5:  *Course name: COMP.2040
6:  *Assignment: PS4b - StringSound
7:  *Instructor's name: Dr. James Daly
8:  *Date: 3/28/22
9:  *Sources Of Help: SFML Website, Professor/Classmate
10: *****/
11:
12: #include <math.h>
13: #include <limits.h>
14:
15: #include <iostream>
16: #include <string>
17: #include <exception>
18: #include <stdexcept>
19: #include <vector>
20:
21: #include "CircularBuffer.h"
22: #include "StringSound.h"
23:
24: #include <SFML/Graphics.hpp>
25: #include <SFML/System.hpp>
26: #include <SFML/Audio.hpp>
27: #include <SFML/Window.hpp>
28:
29: #define CONCERT_A 440.0
30: #define SAMPLES_PER_SEC 44100
31: const int size = 37;
32: const int ascii_limit = 128;
33:
34: std::vector<sf::Int16> makeSamples(StringSound* 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
50:         (sf::VideoMode(300, 200), "SFML Plucked String Sound Guitar Sim");
51:
52:     sf::Event event;
53:     double freq;
54:     std::vector<std::vector<sf::Int16>> Samples(size);
55:     std::vector<sf::SoundBuffer> SoundBuffers(size);
56:     std::vector<sf::Sound> Sounds(size);
57:     std::vector<sf::Int16> sample;
58:
59:     std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
60:     std::unique_ptr<char[]> notes(new char[size]);
61:
62:     int keyboard_len = keyboard.length();
63:     for (int i = 0; i < keyboard_len; i++) {
64:         notes[i] = keyboard[i];
65:     }
```

```
66:
67:     for (int i = 0; i < keyboard_len; i++) {
68:         auto calcFreq = [&](double sound_freq) {
69:             sound_freq = CONCERT_A *
70:                 std::pow(2.0, ((i - 24.0) / 12.0));
71:
72:             return sound_freq;
73:         };
74:
75:         freq = calcFreq(freq);
76:         // std::cout << i << ": " << freq << std::endl;
77:         StringSound S(freq);
78:         sample = makeSamples(&S);
79:         Samples[i] = sample;
80:
81:         if (!SoundBuffers[i].loadFromSamples
82:             (&Samples[i][0], Samples[i].size(), 2, SAMPLES_PER_SEC)) {
83:             throw std::runtime_error
84:                 ("sf::SoundBuffer: failed to load from samples.");
85:         }
86:         Sounds[i].setBuffer(SoundBuffers[i]);
87:     }
88:     char key = 'a';
89:
90:     while (window.isOpen()) {
91:         while (window.pollEvent(event)) {
92:             if (event.type == sf::Event::Closed) {
93:                 window.close();
94:             } else if (event.type == sf::Event::TextEntered) {
95:                 if (event.text.unicode < ascii_limit) {
96:                     key = static_cast<char>(event.text.unicode);
97:
98:                     for (int i = 0; i < keyboard_len; i++) {
99:                         if (key == notes[i]) {
100:                             Sounds[i].play();
101:                         }
102:                     }
103:                 }
104:             }
105:
106:             window.clear();
107:             window.display();
108:         }
109:     }
110:     return 0;
111: }
```

```
1: // Copyright [2022] Tim Truong
2: #pragma once
3: #ifndef CIRCULARBUFFER_H
4: #define CIRCULARBUFFER_H
5:
6: #include <stdint.h>
7: #include <iostream>
8: #include <deque>
9: #include <exception>
10: #include <stdexcept>
11: #include <SFML/System.hpp>
12: #include <SFML/Window.hpp>
13: #include <SFML/Graphics.hpp>
14: #include <SFML/Audio.hpp>
15:
16: class CircularBuffer {
17: public:
18:     explicit CircularBuffer(int capacity);
19:     // create an empty ring buffer, with given max
20:     // capacity
21:     size_t size(); // return number of items currently in the buffer
22:     bool isEmpty(); // is the buffer empty (size equals zero)?
23:     bool isFull(); // is the buffer full (size equals capacity)?
24:     void enqueue(int16_t x); // add item x to the end
25:     int16_t dequeue(); // delete and return item from the front
26:     int16_t peek(); // return (but do not delete) item from the front
27:     int16_t at(int16_t x);
28:     int capacity();
29: private:
30:     std::deque<int16_t> circularBuffer;
31:     int capacity_;
32:     int size_;
33: };
34:
35: #endif
```

```
1: // Copyright [2022] Tim Truong
2:
3: #include "CircularBuffer.h"
4:
5: CircularBuffer::CircularBuffer(int capacity) {
6:     if (capacity < 1) {
7:         throw std::invalid_argument
8:             ("CircularBuffer constructor: capacity must be greater than zero.
");
9:     } else {
10:         capacity_ = capacity;
11:         size_ = 0;
12:         // circularBuffer.resize(capacity_);
13:     }
14: }
15:
16: size_t CircularBuffer::size() {
17:     if (isEmpty()) {
18:         return 0;
19:     } else {
20:         return size_;
21:     }
22: }
23:
24: bool CircularBuffer::isEmpty() {
25:     return size_ == 0;
26: }
27:
28: bool CircularBuffer::isFull() {
29:     auto isBufferFull = [&](int buffer_size) {
30:         if (buffer_size == capacity_) {
31:             return 1;
32:         } else {
33:             return 0;
34:         }
35:     };
36:
37:     bool truth = isBufferFull(size_);
38:
39:     if (truth) {
40:         return 1;
41:     } else {
42:         return 0;
43:     }
44: }
45:
46: void CircularBuffer::enqueue(int16_t x) {
47:     if (isFull()) {
48:         throw std::runtime_error("enqueue: can't enqueue to a full ring."
);
49:     } else {
50:         circularBuffer.push_back(x);
51:         size_++;
52:     }
53:
54:     return;
55: }
56:
57: int16_t CircularBuffer::dequeue() {
58:     if (isEmpty()) {
59:         throw std::runtime_error("dequeue: can't dequeue an empty ring."
);
60:     } else {
61:         int16_t front_ = circularBuffer.front();
62:
```



```
63:         circularBuffer.pop_front();
64:         size_--;
65:
66:         return front_;
67:     }
68: }
69:
70: int16_t CircularBuffer::peek() {
71:     int16_t front_ = circularBuffer.front();
72:     if (isEmpty()) {
73:         throw std::runtime_error("peek: can't peek inside an empty ring."
);
74:     } else {
75:         return front_;
76:     }
77:
78:     return front_;
79: }
80:
81: int16_t CircularBuffer::at(int16_t x) {
82:     return circularBuffer.at(x);
83: }
84:
85: int CircularBuffer::capacity() {
86:     return capacity_;
87: }
```

```
1: // Copyright [2022] Tim Truong
2: #pragma once
3: #ifndef STRINGSOUND_H
4: #define STRINGSOUND_H
5:
6: #include "CircularBuffer.h"
7: #include <math.h>
8: #include <limits.h>
9: #include <vector>
10: #include <cmath>
11: #include <memory>
12: #include <random>
13:
14: class StringSound {
15: public:
16:     explicit StringSound(double frequency);
17:     explicit StringSound(std::vector<sf::Int16> init);
18:     StringSound(const StringSound &obj) = delete; // no copy const
19:     ~StringSound();
20:     void pluck();
21:     void tic();
22:     sf::Int16 sample();
23:     int time();
24: private:
25:     CircularBuffer* _cb;
26:     int _time;
27: };
28:
29: #endif
```

```
1: // Copyright [2022] Tim Truong
2:
3: #include "StringSound.h"
4:
5: const int SAMPLING_RATE = 44100;
6: const double DECAY_RATE = 0.996;
7:
8: StringSound::StringSound(double frequency) {
9:     if (frequency < 1) {
10:         throw std::invalid_argument("Frequency has to be greater than 0")
;
11:     } else {
12:         int size_ = std::ceil(SAMPLING_RATE / frequency);
13:         _cb = new CircularBuffer(size_);
14:
15:         while (!(_cb->isFull())) {
16:             _cb->enqueue(0);
17:         }
18:
19:         _time = 0;
20:     }
21: }
22:
23: StringSound::StringSound(std::vector<sf::Int16> init) {
24:     int size_ = init.size();
25:     if (size_ < 1) {
26:         throw std::invalid_argument("init.size() has to be greater than 0
");
27:     } else {
28:         _cb = new CircularBuffer(size_);
29:         for (auto it = init.begin(); it != init.end(); it++) {
30:             _cb->enqueue(*it);
31:         }
32:         _time = 0;
33:     }
34: }
35:
36: StringSound::~StringSound() {
37:     delete _cb;
38: }
39:
40: void StringSound::pluck() {
41:     while (!(_cb->isEmpty())) {
42:         _cb->dequeue();
43:     }
44:
45:     while (!(_cb->isFull())) {
46:         std::random_device seed;
47:         std::mt19937 r(seed());
48:         std::uniform_int_distribution<sf::Int16> gen(-32768, 32767);
49:
50:         sf::Int16 rand_val = gen(r);
51:         _cb->enqueue((sf::Int16)rand_val);
52:     }
53:
54:     return;
55: }
56:
57: void StringSound::tic() {
58:     auto karplus_result = [](int16_t first, int16_t second) {
59:         float karplus = (DECAY_RATE * 0.5 * (first + second));
60:
61:         return karplus;
62:     };
63:
```

```
64:     int16_t first_sample = _cb->dequeue();
65:     int16_t second_sample = _cb->peek();
66:
67:     sf::Int16 karplus = karplus_result(first_sample, second_sample);
68:
69:     _cb->enqueue(karplus);
70:
71:     _time++;
72:
73:     return;
74: }
75:
76: sf::Int16 StringSound::sample() {
77:     return (sf::Int16)_cb->peek();
78: }
79:
80: int StringSound::time() {
81:     return _time;
82: }
```

```
1: // Copyright [2022] Tim Truong
2:
3: #define BOOST_TEST_DYN_LINK
4: #define BOOST_TEST_MODULE Main
5: #include <boost/test/unit_test.hpp>
6: #include "CircularBuffer.h"
7: #include "StringSound.h"
8:
9: BOOST_AUTO_TEST_CASE(SS_Constructor) {
10:     BOOST_REQUIRE_NO_THROW(StringSound S(10));
11:     BOOST_REQUIRE_NO_THROW(StringSound S2(150));
12:     BOOST_REQUIRE_NO_THROW(StringSound S3(162));
13:     BOOST_REQUIRE_NO_THROW(StringSound S4(11));
14:
15:     BOOST_REQUIRE_THROW(StringSound S5(0), std::exception);
16:     BOOST_REQUIRE_THROW(StringSound S6(0), std::invalid_argument);
17:     BOOST_REQUIRE_THROW(StringSound S7(0), std::invalid_argument);
18:
19:     std::vector<sf::Int16> v;
20:
21:     BOOST_REQUIRE_THROW(StringSound S8(v), std::exception);
22:     BOOST_REQUIRE_THROW(StringSound S9(v), std::invalid_argument);
23:     BOOST_REQUIRE_THROW(StringSound S10(v), std::invalid_argument);
24: }
25:
26: BOOST_AUTO_TEST_CASE(SS_Sample_Tic_Time) {
27:     std::vector<sf::Int16> v;
28:
29:     v.push_back(100);
30:     v.push_back(500);
31:     v.push_back(600);
32:     v.push_back(-2000);
33:
34:     BOOST_REQUIRE_NO_THROW(StringSound S_T(v));
35:
36:     StringSound S(v);
37:
38:     BOOST_REQUIRE(S.sample() == 100);
39:
40:     S.tic();
41:     BOOST_REQUIRE(S.sample() == 500);
42:
43:     S.tic();
44:     BOOST_REQUIRE(S.sample() == 600);
45:
46:     S.tic();
47:     BOOST_REQUIRE(S.sample() == -2000);
48:
49:     S.tic();
50:     BOOST_REQUIRE(S.sample() == 298);
51:
52:     S.tic();
53:     BOOST_REQUIRE(S.sample() == 547);
54:
55:     BOOST_REQUIRE(S.time() == 5);
56: }
57:
58: BOOST_AUTO_TEST_CASE(Buffer_Constructor) {
59:     BOOST_REQUIRE_NO_THROW(CircularBuffer B(1287));
60:     BOOST_REQUIRE_NO_THROW(CircularBuffer B2(3));
61:     BOOST_REQUIRE_NO_THROW(CircularBuffer B3(1));
62:
63:     BOOST_REQUIRE_THROW(CircularBuffer B4(0), std::exception);
64:     BOOST_REQUIRE_THROW(CircularBuffer B5(0), std::invalid_argument);
65:     BOOST_REQUIRE_THROW(CircularBuffer B6(-1), std::invalid_argument);
```

```
66: }
67:
68: BOOST_AUTO_TEST_CASE(Buffer_Full_Empty) {
69:     CircularBuffer B(3);
70:     BOOST_REQUIRE(B.isEmpty() == 1);
71:
72:     CircularBuffer B2(3);
73:     B2.enqueue(2);
74:     BOOST_REQUIRE(B2.isEmpty() == 0);
75:
76:     CircularBuffer B3(3);
77:     B3.enqueue(1);
78:     B3.enqueue(2);
79:     B3.enqueue(3);
80:     BOOST_REQUIRE(B3.isFull() == 1);
81:
82:     CircularBuffer B4(3);
83:     B4.enqueue(7);
84:     BOOST_REQUIRE(B4.isFull() == 0);
85: }
86:
87: BOOST_AUTO_TEST_CASE(Buffer_Peek_Size) {
88:     CircularBuffer B(3);
89:     B.enqueue(5);
90:     B.enqueue(6);
91:     BOOST_REQUIRE(B.size() == 2);
92:
93:     CircularBuffer B2(4);
94:     B2.enqueue(9);
95:     B2.enqueue(7);
96:     BOOST_REQUIRE(B2.peek() == 9);
97:
98:     CircularBuffer B3(3);
99:     BOOST_REQUIRE_THROW(B3.peek(), std::runtime_error);
100: }
101:
102: BOOST_AUTO_TEST_CASE(Buffer_Enqueue_Dequeue) {
103:     CircularBuffer B(7);
104:     B.enqueue(3);
105:     BOOST_REQUIRE(B.peek() == 3);
106:     B.enqueue(7);
107:     BOOST_REQUIRE(B.peek() == 3);
108:
109:     CircularBuffer B2(5);
110:     for (int i = 0; i < 5; i++) {
111:         BOOST_REQUIRE_NO_THROW(B2.enqueue(i));
112:     }
113:
114:     BOOST_REQUIRE_THROW(B2.enqueue(6), std::runtime_error);
115:
116:     CircularBuffer B3(7);
117:     B3.enqueue(4);
118:     B3.enqueue(6);
119:     B3.enqueue(9);
120:     B3.dequeue();
121:     BOOST_REQUIRE(B3.size() == 2);
122:     BOOST_REQUIRE(B3.peek() == 6);
123:
124:     CircularBuffer B4(5);
125:     BOOST_REQUIRE_NO_THROW(B4.enqueue(0));
126:     BOOST_REQUIRE_NO_THROW(B4.enqueue(1));
127:     BOOST_REQUIRE_NO_THROW(B4.enqueue(2));
128:     BOOST_REQUIRE_NO_THROW(B4.enqueue(3));
129:     BOOST_REQUIRE_NO_THROW(B4.enqueue(4));
130:
```

```
131: BOOST_REQUIRE(B4.dequeue() == 0);
132: BOOST_REQUIRE(B4.dequeue() == 1);
133: BOOST_REQUIRE(B4.dequeue() == 2);
134: BOOST_REQUIRE(B4.dequeue() == 3);
135: BOOST_REQUIRE(B4.dequeue() == 4);
136:
137: BOOST_REQUIRE_THROW(B4.dequeue(), std::runtime_error);
138:
139: CircularBuffer B5(3);
140: BOOST_REQUIRE_NO_THROW(B5.enqueue(1));
141: BOOST_REQUIRE_NO_THROW(B5.enqueue(2));
142: BOOST_REQUIRE_NO_THROW(B5.enqueue(3));
143:
144: BOOST_REQUIRE(B5.dequeue() == 1);
145: BOOST_REQUIRE(B5.dequeue() == 2);
146: BOOST_REQUIRE(B5.dequeue() == 3);
147:
148: BOOST_REQUIRE_NO_THROW(B5.enqueue(7));
149: BOOST_REQUIRE_NO_THROW(B5.enqueue(8));
150: BOOST_REQUIRE_NO_THROW(B5.enqueue(9));
151:
152: BOOST_REQUIRE(B5.peek() == 7);
153: }
154:
155: BOOST_AUTO_TEST_CASE(Buffer_Lambda) {
156:     auto vecSum = [](size_t amount, CircularBuffer Buff) {
157:         CircularBuffer B = Buff;
158:
159:         int16_t sum = 0;
160:         for (size_t i = 0; i < amount; i++) {
161:             sum = sum + B.at(i);
162:         }
163:         return sum;
164:     };
165:
166:     int capacity = 5;
167:     CircularBuffer B(capacity);
168:     for (int i = 0; i < capacity; i++) {
169:         B.enqueue(i);
170:     }
171:     int16_t sum = vecSum(capacity, B);
172:     int16_t val = 10;
173:     BOOST_REQUIRE_EQUAL(sum, val);
174: }
```