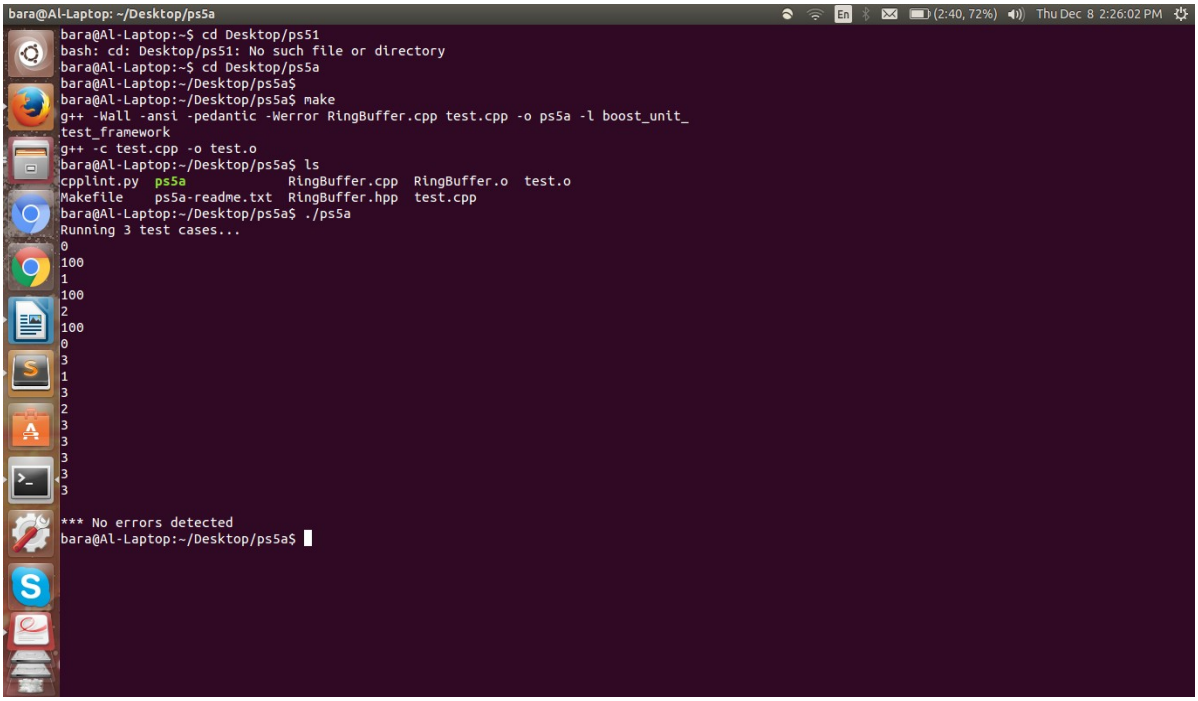
PS5: Ringer Buffer and Guitar Hero

PS5a:

In this assignment, we needed implement the ring buffer that will hold the guitar string position data, and write test functions and exception handling. This assignment was to create the length of the string determining its fundamental frequency of vibration. To accomplish this task, I needed to create a ring buffer that will determine the fundamental frequency and the harmonics of the frequency.

The algorithms we used was a system called the cyclic wrap- around. This concept was to basically reuse an array without extending the amount of elements. I needed to have 2 pointers called the first and last arrays. Each time a user would enter a number in array, the pointer last would place the number in the array and go to the next one. When the user wants to takes out the first array, the first pointer would remove the first number and save the address of the next element. If the queue was full, I could set the pointer last to the beginning of the array. If it was full, it would notify the user its full and stop. Until the user dequeues the the value, the user can not enter any more numbers. If dequeue is continued to be called and the array is empty, the user will be notified that it is empty by throwing a exception.

The understanding of an array that wraps around to replace a element, it a great concept that could save memory in the future. It can be used to be efficient with memory. Whats even more great is that I figured out that members in a class can be accessed with needed gets and sets. Even though I should have known this before, figuring it out in this program helped out a lot. I also learned about throwing a exception when a error has occurred.



(Compiled with Makefile)

```
    1: cc = g++
    2:
    3: all : RingBuffer test
    4:
    5: RingBuffer : RingBuffer.o test.o
    6:         $(cc) -Wall -ansi -pedantic -Werror RingBuffer.cpp test.cpp -o ps5a
-l boost_unit_test_framework
    7:
    8: RingBuffer.o : RingBuffer.hpp
    9:         $(cc) -c RingBuffer.cpp -o RingBuffer.o
   10:
   11: test : test.cpp RingBuffer.hpp
   12:         $(cc) -c test.cpp -o test.o
   13:
   14:
   15: clean :
   16:         rm  *.o ps5a
```

```
 1: /*Copyright [2016] <Albara Mehene> */
 2: /*#define BOOST_TEST_DYN_LINK
 3: #define BOOST_TEST_MODULE Main
 4: */
 5:
 6: #define BOOST_TEST_DYN_LINK
 7: #define BOOST_TEST_MODULE Main
 8: #include <boost/test/unit_test.hpp>
 9:
10: #include <stdint.h>
11: #include <iostream>
12: #include <string>
13: #include <exception>
14: #include <stdexcept>
15:
16: #include "RingBuffer.hpp"
17:
18: BOOST_AUTO_TEST_CASE(RBcontructor) {
19:     // normal constructor
20:     BOOST_REQUIRE_NO_THROW(RingBuffer(100));
21:
22:     // this should fail
23:     BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
24:     BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
25: }
26:
27: BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
28:     RingBuffer rb(100);
29:
30:     rb.enqueue(2);
31:     rb.enqueue(1);
32:     rb.enqueue(0);
33:
34:     BOOST_REQUIRE(rb.dequeue() == 2);
35:     BOOST_REQUIRE(rb.dequeue() == 1);
36:     BOOST_REQUIRE(rb.dequeue() == 0);
37:
38:     BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
39: }
40:
41: BOOST_AUTO_TEST_CASE(test1) {
42:     RingBuffer rb(3);
43:     rb.enqueue(2);
44:     rb.enqueue(3);
45:     rb.enqueue(4);
46:     BOOST_REQUIRE(rb.isFull());
47:     BOOST_REQUIRE_THROW(rb.enqueue(5), std::runtime_error)
48: }
```

```
 1: /*Copyright [2016] <Albara Mehene> */
 2:
 3: #include "RingBuffer.hpp"
 4:
 5: // creates a empty ringbuffer with a max capacity
 6: RingBuffer::RingBuffer(int capacity) {
 7: if(capacity < 1) {
 8:     throw std::invalid_argument("capacity must be greater than zero");
 9: }
10:     cap = capacity;
11:     count = 0;
12:     array = new int16_t[capacity];
13:     first = array;
14:     last = array;
15: }
16:
17: // returns the number of items currently in the buffer
18: int RingBuffer::size() {
19:     return count;
20: }
21:
22: // checks to see if the buffer is empty
23: bool RingBuffer::isEmpty() {
24:     if (count == 0) {
25:         return 1;
26:     } else {
27:         return 0;
28:     }
29: }
30:
31: // checks to see if the buffer is full
32: bool RingBuffer::isFull() {
33:     if (count == cap) {
34:        return 1;
35:     } else {
36:        return 0;
37:     }
38: }
39:
40: // add item x to the end of buffer
41: void RingBuffer::enqueue(int16_t x) {
42: if(isFull() == 1) {
43:     throw std::runtime_error("can't enqueue to a full ring");
44: }
45:
46:     if (last == (array+(cap-1))){
47:          count++;
48:          (*last) = x;
49:          last = array;
50:     }else{
51:         count++;
52:         (*last) = x;
53:         last = (last + 1);
54:     }
55: }
56:
57: // deletes and returns the item from the front of the buffer
58: int16_t RingBuffer::dequeue() {
59:     if (isEmpty() == 1) {
60:        throw std::runtime_error("can't dequeue to a empty ring");
61:     }
```

```
62:
63:     int16_t store;
64:
65:     if(first == (array+(cap-1))){
66:       count--;
67:       store = (*first);
68:       first = array;
69:       return store;
70:     }else{
71:         count--;
72:         store = (*first);
73:         first = (first+1);
74:         return store;
75:     }
76: }
77:
78: // returns item from the front without deleting it
79: int16_t RingBuffer::peek() {
80:     int16_t temp = (*first);
81:     return temp;
82: }
83:
84: RingBuffer::~RingBuffer() {
85:     delete[] array;
86: }
87: void RingBuffer::print_out(){
88:
89:     for(int i = 0; i < cap;i++){
90:         std::cout << "Array:   " << array[i] << std::endl;
91:     }
92: }
```

```
 1: /*Copyright [2016] <Albara Mehene> */
 2: #ifndef RINGBUFFER_HPP
 3: #define RINGBUFFER_HPP
 4:
 5: #include <boost/test/unit_test.hpp>
 6:
 7: #include <SFML/Graphics.hpp>
 8: #include <SFML/System.hpp>
 9: #include <SFML/Audio.hpp>
10: #include <SFML/Window.hpp>
11: #include <stdint.h>
12: #include <iostream>
13: #include <string>
14: #include <exception>
15: #include <stdexcept>
16: #include <vector>
17:
18: class RingBuffer{
19:  public:
20:     explicit RingBuffer(int capacity);
21:     int size();
22:     bool isEmpty();
23:     bool isFull();
24:     void enqueue(int16_t x);
25:     int16_t dequeue();
26:     int16_t peek();
27:     void print_out();
28:     ~RingBuffer();
29:  private:
30:     int16_t *first;
31:     int16_t *last;
32:     int16_t *array;
33:     int count;
34:     int cap;
35: };
36:
37: #endif
```