

# Computing IV Project Portfolio Stephanie La Fall 2020

## Contents

<b>1</b>	<b>PS0 - SFML Hello World Project</b>	<b>4</b>
1.1	Overview . . . . .	4
1.2	Key Concepts . . . . .	4
1.3	Learnings and Accomplishments . . . . .	4
1.4	main.cpp . . . . .	4
1.5	Output . . . . .	6
1.6	screenshot.png . . . . .	6
1.7	sprite.png . . . . .	7
<b>2</b>	<b>Ps1a - Linear Feedback Shift Register Part A</b>	<b>7</b>
2.1	Overview . . . . .	7
2.2	Key Concepts . . . . .	7
2.3	Makefile . . . . .	8
2.4	Learnings and Accomplishments . . . . .	8
2.5	test.cpp . . . . .	8
2.6	FibLFSR.h . . . . .	9
2.7	FibLFSR.cpp . . . . .	10
<b>3</b>	<b>Ps1b - Linear Feedback Shift Register Part B</b>	<b>12</b>
3.1	Overview . . . . .	12
3.2	Key Concepts . . . . .	12
3.3	Learnings and Accomplishments . . . . .	12
3.4	Makefile . . . . .	12
3.5	PhotoMagic.cpp . . . . .	13
3.6	LFSR.h . . . . .	15
3.7	LFSR.cpp . . . . .	16
3.8	encode.png . . . . .	18
3.9	decode.png . . . . .	19
<b>4</b>	<b>Ps2a - N-Body Simulation Part A</b>	<b>19</b>
4.1	Overview . . . . .	19
4.2	Key Concepts . . . . .	20
4.3	Learnings and Accomplishments . . . . .	20
4.4	Makefile . . . . .	20
4.5	main.cpp . . . . .	21
4.6	NBody.h . . . . .	22
4.7	NBody.cpp . . . . .	23
4.8	screenshot.png . . . . .	27

<b>5</b>	<b>Ps2b - N-Body Simulation Part B</b>	<b>28</b>
5.1	Overview . . . . .	28
5.2	Key Concepts . . . . .	28
5.3	Learnings and Accomplishments . . . . .	28
5.4	Makefile . . . . .	28
5.5	main.cpp . . . . .	29
5.6	NBody.h . . . . .	30
5.7	NBody.cpp . . . . .	32
5.8	Screenshot of Planets . . . . .	38
<b>6</b>	<b>Ps3a - Synthesizing a Plucked String Sound Part A</b>	<b>38</b>
6.1	Overview . . . . .	38
6.2	Key Concepts . . . . .	39
6.3	Learnings and Accomplishments . . . . .	39
6.4	Makefile . . . . .	39
6.5	test.cpp . . . . .	39
6.6	CircularBuffer.h . . . . .	41
6.7	CircularBuffer.cpp . . . . .	42
<b>7</b>	<b>Ps3b - Synthesizing a Plucked String Sound Part B</b>	<b>43</b>
7.1	Overview . . . . .	43
7.2	Key Concepts . . . . .	43
7.3	Learnings and Accomplishments . . . . .	43
7.4	Makefile . . . . .	43
7.5	KSGuitarSim.cpp . . . . .	44
7.6	CircleBuffer.h . . . . .	46
7.7	StringSound.h . . . . .	47
7.8	CircleBuffer.cpp . . . . .	48
7.9	Stringsound.cpp . . . . .	50
7.10	Sound result . . . . .	53
<b>8</b>	<b>Ps4 - DNA Sequence Alignment</b>	<b>53</b>
8.1	Overview . . . . .	53
8.2	Key Concepts . . . . .	53
8.3	Learnings and Accomplishments . . . . .	53
8.4	Makefile . . . . .	54
8.5	main.cpp . . . . .	55
8.6	EditDistance.h . . . . .	56
8.7	EditDistance.cpp . . . . .	57
8.8	Execution Result . . . . .	60
<b>9</b>	<b>Ps5 - Markov Model of Natural Language</b>	<b>60</b>
9.1	Overview . . . . .	60
9.2	Key Concepts . . . . .	60
9.3	Learnings and Accomplishments . . . . .	61
9.4	Makefile . . . . .	61
9.5	MModel.h . . . . .	62
9.6	MModel.cpp . . . . .	63

<b>10 Ps6 - Kronos Time Clock: Introduction to Regular Expression Parsing</b>	<b>65</b>
10.1 Overview . . . . .	65
10.2 Key Concepts . . . . .	65
10.3 Learnings and Accomplishments . . . . .	66
10.4 Makefile . . . . .	66
10.5 main.cpp . . . . .	66
10.6 Rpt Files . . . . .	69

# 1 PS0 - SFML Hello World Project

## 1.1 Overview

This project aimed to introduce us to the SFML library for graphics. We had to create a simple application that featured a sprite that moved and reacted to user input.

First, I rendered a window, made a texture object, then loaded a file called sprite.png from a file to the texture. From there, I made a sprite object, and drew the sprite to the screen. Based on the user's input, the keys left, right, up, down would move the sprite in the corresponding direction. While the window was open, I also incorporated the ability to rotate the image.

## 1.2 Key Concepts

The key concepts introduced in this assignment were the SFML draw loop, SFML event handling, and the basics of sprites in the SFML library.

## 1.3 Learnings and Accomplishments

I had a bit of experience with SFML from Computing III since my professor introduced it to us in the past semester, so the main thing I learned about for this assignment was how to use sprites and how to move them in SFML since I never used them before.

## 1.4 main.cpp

```
1 //Stephanie La
2 #include <SFML/Graphics.hpp>
3
4 int main()
5 {
6     sf::RenderWindow window(sf::VideoMode(1000, 1000), "SFML window!");
7     window.setFramerateLimit(90); // call it once, after creating
        the window
8     //load a sprite to display
9     sf::Texture texture;
10    if(!texture.loadFromFile("sprite.png")){
11        return EXIT_FAILURE;
12    }
13    sf::Sprite sprite(texture);
14    float position1 = 500; //left and right
15    float position2 = 500; //up and down
16    while (window.isOpen())
17    {
18        sf::Event event;
19        while (window.pollEvent(event))
20        {
21            if (event.type == sf::Event::Closed)
22                window.close();
23            //define arrow keys
```

```

24         //use the key pressed/key release
25     }
26     sprite.setPosition(position1, position2);
27     if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)){
28         position1--;
29     }
30     if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
31         position1++;
32     }
33     if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up)){
34         position2--;
35     }
36     if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down)){
37         position2++;
38     }
39     sprite.rotate(9.f);
40     window.clear();
41     //draw sprite
42     window.draw(sprite);
43     window.display();
44 }
45
46 return 0;
47 }

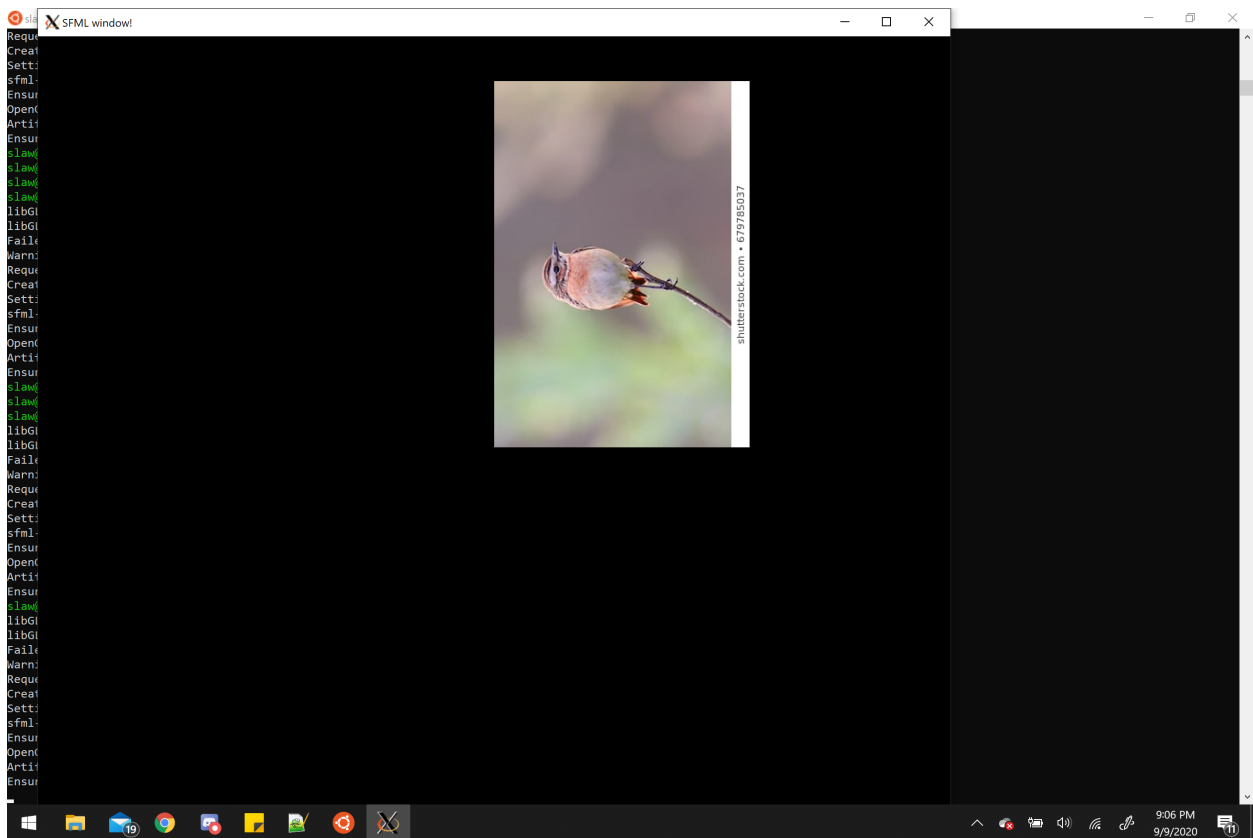
```

## 1.5 Output

I created an image of a bird that would move up,down, left, or right and would rotate while moving.

## 1.6 screenshot.png

Figure 1: Screenshot of Rotating Sprite Sideways



## 1.7 sprite.png

Figure 2: Sprite Image



shutterstock.com • 679785037

## 2 Ps1a - Linear Feedback Shift Register Part A

### 2.1 Overview

This project consisted of parts A and B. In this first half of the assignment, we had to create a program with a linear feedback shift register (LFSR) and then use it to implement a simple form of encryption and decryption for digital pictures. We had to implement the FibLFSR class and implement unit tests using the Boost test framework. I used a vector of integers to hold as many bits as needed.

### 2.2 Key Concepts

One key concept introduced in this assignment was how to use the Boost Unit Testing library to create tests for your program and automatically run them on your code so that you can ensure it is working correctly without any errors.

## 2.3 Makefile

Another key concept introduced in this assignment was how to make Makefiles with flags and targets. I had to understand what an executable was, and then go off of it: what files did I need, what flags do I need to compile the program.

```
1 CC = g++
2 CFLAGS = -g -w -Wall -Werror -ansi -pedantic
3
4 ps1a: FibLFSR.o test.o
5     $(CC) $(CFLAGS) -o ps1a FibLFSR.o test.o -
        lboost_unit_test_framework
6 all: ps1a
7 FibLFSR.o: FibLFSR.cpp FibLFSR.h
8     $(CC) $(CFLAGS) -c FibLFSR.cpp
9 test.o: test.cpp
10    $(CC) $(CFLAGS) -c test.cpp
11 clean:
12    rm ps1a
```

## 2.4 Learnings and Accomplishments

This was the first time I have ever used Boost libraries to test my code. I had previously used test functions in Computing II to test my Evil Hangman project, but this was my first time writing my own test functions.

## 2.5 test.cpp

I decided to make a boost test function to test out if the step function would return the following results of 0 or 1, which was successful. Another boost test was made to test the bytes will results in zeros also using the step() function, since the step function was the function doing the work.

```
1 // Dr. Rykalova
2 // test.cpp for PS1a
3 // updated 1/31/2020
4
5 #include <iostream>
6 #include <string>
7
8 #include "FibLFSR.h"
9
10 #define BOOST_TEST_DYN_LINK
11 #define BOOST_TEST_MODULE Main
12 #include <boost/test/unit_test.hpp>
13
14
15 BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
16
17     FibLFSR l("1011011000110110");
18     BOOST_REQUIRE(l.step() == 0);
```



```

19 BOOST_REQUIRE(l.step() == 0);
20 BOOST_REQUIRE(l.step() == 0);
21 BOOST_REQUIRE(l.step() == 1);
22 BOOST_REQUIRE(l.step() == 1);
23 BOOST_REQUIRE(l.step() == 0);
24 BOOST_REQUIRE(l.step() == 0);
25 BOOST_REQUIRE(l.step() == 1);
26
27 FibLFSR l2("1011011000110110");
28 BOOST_REQUIRE(l2.generate(9) == 51);
29 }
30 //test if all the bits will result in zeros using step functions
   worked
31 BOOST_AUTO_TEST_CASE(allBitsOne) {
32
33     FibLFSR FibLFSR_One("1111111111111111");
34     BOOST_REQUIRE(FibLFSR_One.step() == 0);
35     BOOST_REQUIRE(FibLFSR_One.step() == 0);
36     BOOST_REQUIRE(FibLFSR_One.step() == 0);
37     BOOST_REQUIRE(FibLFSR_One.step() == 0);
38     BOOST_REQUIRE(FibLFSR_One.step() == 0);
39     BOOST_REQUIRE(FibLFSR_One.step() == 0);
40     BOOST_REQUIRE(FibLFSR_One.step() == 0);
41     BOOST_REQUIRE(FibLFSR_One.step() == 0);
42
43 }
44
45 //this test was to check if the xor functions implemented in step
   worked
46 BOOST_AUTO_TEST_CASE(sixteenBits)
47 {
48     FibLFSR line("1101001101010001");
49     BOOST_REQUIRE(line.step() == 0);
50     BOOST_REQUIRE(line.step() == 1);
51     BOOST_REQUIRE(line.step() == 1);
52     BOOST_REQUIRE(line.step() == 0);
53     BOOST_REQUIRE(line.step() == 1);
54 }

```

## 2.6 FibLFSR.h

In my header file,I decided to use a vector of integers to hold as many bits as needed, and it would automatically resize to fit more data. I had also created variable integers tap10, tap12,tap13, and tap 15 in order to save a tap and use that to compare to the other taps. I had also created a string s in order to store my result.

```

1 #ifndef FibLFSR_h
2 #define FibLFSR_h

```

```

3 #include <iostream>
4 #include <string>
5 #include <vector>
6
7 class FibLFSR {
8 public:
9     FibLFSR(std::string seed); // constructor to create LFSR with
10                                // the given initial seed and
11                                tap
12
13     int step();                // simulate one step and return the
14                                // new bit as 0 or 1
15
16     int generate(int k);       // simulate k steps and return
17                                // k-bit integer
18
19     ~FibLFSR();
20     friend std::ostream& operator<<(std::ostream& out, const FibLFSR
21                                     & fibLFSR);
22
23 private:
24     std::string s;
25     std::vector<int> data;
26     int tap10;
27     int tap12;
28     int tap13;
29     int tap15; //end of the 16 bits
30
31 };
32 #endif //FibLFSR_h

```

## 2.7 FibLFSR.cpp

The step function simulates one step of LFSR and returns to the rightmost bit. The generate function would return a k-bit integer simulating k steps, where the bit would be doubled and added to the bit returned from step. I did not use an xor for my step function, but I saved the result between two taps into a variable, and then tested if the other taps were equal to the variable. In the step() function, I knew strings were read from left to right, so I had to reverse the seed so the bits would be 2,3, and 5 from left to right.

```

1 //Stephanie La
2 #include <iostream>
3 #include "FibLFSR.h"
4 using namespace std;
5
6
7 FibLFSR::FibLFSR(string seed) {
8     while (seed.length() != 16) {

```

```

9         cout << "The sequence entered is less or greater than 16.
           Enter another sequence:";
10        cin >> seed;
11    }
12    s = seed; //initializing seed string
13 }
14
15 int FibLFSR::step() {
16     int result = 0;
17     //srings are read in left to right, so 10,12,13 becomes 2,3,5
        left to right
18     tap13 = s.at(2) - '0'; //subtract the ascii value from 2
19     tap12 = s.at(3) - '0';
20     tap10 = s.at(5) - '0';
21     tap15 = s.at(0) - '0';
22
23     //if they both equal 1 or 0, output is 0
24     //ex. 0 xor 0 = 0 and 1 xor 1 = 0
25     if ((tap15 == 1 && tap13 == 1) || (tap15 == 0 && tap13 == 0)) {
26         result = 0;
27     }
28     else {
29         result = 1;
30     }
31     if (result == tap12) {
32         result = 0;
33     }
34     else {
35         result = 1;
36     }
37     if (result == tap10) {
38         result = 0;
39     }
40     else {
41         result = 1;
42     }
43     s.erase(s.begin()); //erase the bit spot at the beginning
44     s.push_back(result + '0'); //push back output towards the back
45     return result;
46 }
47
48 int FibLFSR::generate(int k) {
49     int x = 0;
50     for (int i = 0; i < k; i++) {
51         x = x * 2;
52         x += step();
53     }
54     return x;

```

```

55 }
56
57 FibLFSR::~FibLFSR() {
58     data.clear();
59 }
60
61 ostream& operator<<(std::ostream& out, const FibLFSR& fibLFSR) {
62     return out << fibLFSR.s;
63 }

```

## 3 Ps1b - Linear Feedback Shift Register Part B

### 3.1 Overview

This project consisted of parts A and B. In this second half of the assignment, we had to create a program to read three arguments from the command line: source image filename, output image filename, and the FibLFSR seed. We had to use SFML to load the source image and display it in its own window side by side with the encode and decode images.

### 3.2 Key Concepts

We needed to learn about the properties of the XOR function in order to learn how each pixel would be encrypted. We also had to learn about how to handle images in SFML and how to have multiple windows open at the same time in a program.

### 3.3 Learnings and Accomplishments

Before this project, I had never used SFML to alter a photo, or encrypt a photo. This was an entirely new concept for me, where cryptography reminded me of the many techniques people in cybersecurity utilize. It was also my first time where I had to figure out how to use SFML to open two windows, versus when I had always opened one.

### 3.4 Makefile

The makefile was the same format, with the additional libraries graphics, window, and system used from the SFML library.

```

1 CC = g++
2 CFLAGS = -g -w -Wall -Werror -ansi -pedantic
3 LDLIBS = -lsfml-graphics -lsfml-window -lsfml-system
4
5 Photomagic: LFSR.o PhotoMagic.o
6     $(CC) $(CFLAGS) -o Photomagic LFSR.o PhotoMagic.o $(LDLIBS)
7 all: Photomagic
8 FibLFSR.o: LFSR.cpp LFSR.h
9     $(CC) $(CFLAGS) -c LFSR.cpp
10 PhotoMagic.o: PhotoMagic.cpp
11     $(CC) $(CFLAGS) -c PhotoMagic.cpp
12

```

```

13 clean:
14     rm *.o Photomagic

```

### 3.5 PhotoMagic.cpp

In the PhotoMagic.cpp file, I made a window, loaded the image from file and then made a sprite object. Then, I looped through the size of each x and y coordinates and implemented the xor function with each bit dereferencing the generate() function. After the loop, I made second window where the output would be displayed. While the windows were open, I would draw the sprite inputs and outputs into the respective windows. This file should command line arguments:

Figure 3: Command Line Arguments

```
% PhotoMagic input-file.png output-file.png 1011011000110110
```

In the main function, I assigned each part of the argv[] array to each order of the commands. They were executed in the order they were given. At the end, the encoded and decoded images are displayed in its own windows.

```

1  #include <iostream>
2  #include <string>
3  #include "LFSR.h"
4  #include <SFML/Graphics.hpp>
5
6  using namespace std;
7  void transform(sf::Image& image, FibLFSR* bit) {
8
9      /*if (!image.loadFromFile("picture.png"))
10         exit(1);*/
11
12     sf::Vector2u size = image.getSize();
13     sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Mario
14         Mushroom Input");
15
16     sf::Texture texture_input;
17     texture_input.loadFromImage(image);
18
19     sf::Sprite sprite_input;
20     sprite_input.setTexture(texture_input);
21
22     // p is a pixel
23     sf::Color p;
24     int x_size = image.getSize().x;
25     int y_size = image.getSize().y;
26     //manipulate image
27     // create photographic negative image
28     for (int x = 0; x < x_size; x++) {

```

```

29         p = image.getPixel(x, y);
30         p.r = p.r ^ bit->generate(8);
31         p.g = p.g ^ bit->generate(8);
32         p.b = p.b ^ bit->generate(8);
33         image.setPixel(x, y, p);
34     }
35 }
36
37 sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "Mario
    Mushroom output");
38
39 sf::Texture texture_output;
40 texture_output.loadFromImage(image);
41
42 sf::Sprite sprite_output;
43 sprite_output.setTexture(texture_output);
44
45 while (window1.isOpen() && window2.isOpen()) {
46     sf::Event event;
47     while (window1.pollEvent(event)) {
48         if (event.type == sf::Event::Closed)
49             window1.close();
50     }
51     while (window2.pollEvent(event)) {
52         if (event.type == sf::Event::Closed)
53             window2.close();
54     }
55     window1.clear();
56     window1.draw(sprite_input);
57     window1.display();
58     window2.clear();
59     window2.draw(sprite_output);
60     window2.display();
61 }
62 }
63
64
65
66
67 int main(int argc, char** argv) {
68     argv[0] = "./Photomagic";//executable
69     FibLFSR bit(argv[3]);// seed 1011011000110110
70     sf::Image image;
71     //image.loadFromFile(argv[1]);
72     if (!image.loadFromFile (argv[1])) //picture.png
73         exit(1);
74     transform(image, &bit);
75     //image.saveToFile(argv[2]);

```

```

76     if (!image.saveToFile(argv[2])) //picture-out.png
77         exit(1);
78
79     return 0;
80 }

```

### 3.6 LFSR.h

In the LFSR header file, I did not make any big changes, and utilized the same classes for my cpp file to make the function definitions.

```

1  #ifndef FibLFSR_h
2  #define FibLFSR_h
3  #include <SFML/System.hpp>
4  #include <SFML/Window.hpp>
5  #include <SFML/Graphics.hpp>
6  #include <iostream>
7  #include <string>
8  #include <vector>
9  using namespace std;
10
11 class FibLFSR {
12 public:
13     FibLFSR(std::string seed);
14     int step();
15     int generate(int k);
16     ~FibLFSR();
17     friend std::ostream& operator<<(std::ostream& out, const FibLFSR
        & fibLFSR);
18
19 private:
20     int length;
21     std::string s;
22     std::vector<int>data;
23     bool isBinary(std::string seed); //to check if string is a
        binary password
24     int tap10;
25     int tap12;
26     int tap13;
27     int tap15; //end of the 16 bits
28
29 };
30 #endif //FibLFSR_h
31
32 inline void transform(sf::Image& image, FibLFSR* bit);

```

### 3.7 LFSR.cpp

In the LFSR cpp file, I left everything as the same as it was in part A, and Incorporated the transform function into the step function to extract the red, green, and blue components of the color, and XOR the red, green, and blue components with a newly-generated 8-bit integer. Then a new color is created and set to the pixel in the new picture to that color.

```
1 //Stephanie La
2 #include "LFSR.h"
3 #include <iostream>
4 using namespace std;
5
6
7 FibLFSR::FibLFSR(std::string seed) {
8     while (seed.length() != 16) {
9         cout << "The sequence entered is less or greater than 16.
10             Enter another sequence:";
11         cin >> seed;
12     }
13     s = seed; //initializing seed string
14 }
15 int FibLFSR::step() {
16     int result = 0;
17     //srings are read in left to right, so 10,12,13 becomes 2,3,5
18     //left to right
19     tap13 = s.at(2) - '0'; //subtract the ascii value from 2
20     tap12 = s.at(3) - '0';
21     tap10 = s.at(5) - '0';
22     tap15 = s.at(0) - '0';
23
24     //if they both equal 1 or 0, output is 0
25     //ex. 0 xor 0 = 0 and 1 xor 1 = 0
26     if ((tap15 == 1 && tap13 == 1) || (tap15 == 0 && tap13 == 0)) {
27         result = 0;
28     }
29     else {
30         result = 1;
31     }
32     if (result == tap12) {
33         result = 0;
34     }
35     else {
36         result = 1;
37     }
38     if (result == tap10) {
39         result = 0;
40     }
41     else {
```



```

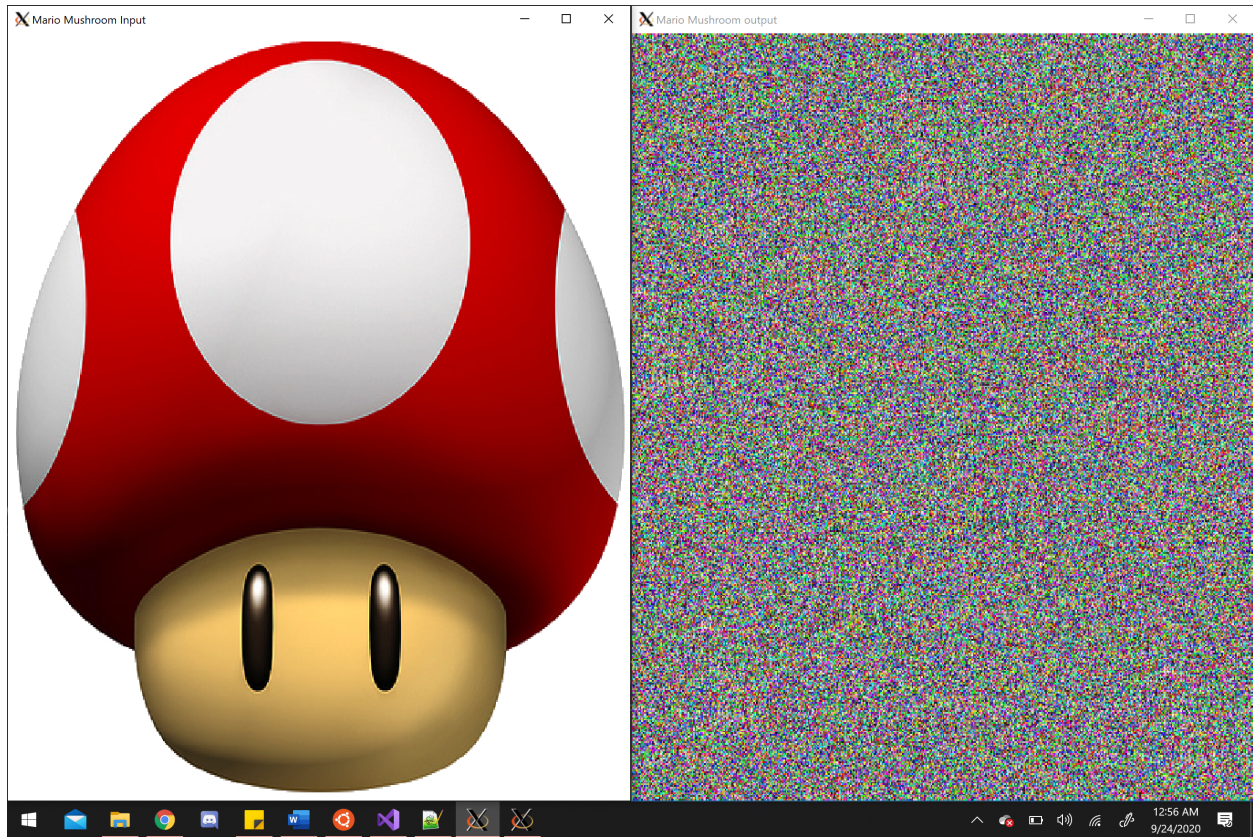
41         result = 1;
42     }
43     s.erase(s.begin()); //erase the bit spot at the beginning
44     s.push_back(result + '0'); //push back output towards the back
45     return result;
46 }
47
48 int FibLFSR::generate(int k) {
49     int x = 0;
50     for (int i = 0; i < k; i++) {
51         x = x * 2;
52         x += step();
53     }
54     return x;
55 }
56
57 FibLFSR::~FibLFSR() {
58     data.clear();
59 }
60
61 ostream& operator<<(std::ostream& out, const FibLFSR& fibLFSR) {
62     return out << fibLFSR.s;
63 }

```

### 3.8 encode.png

On the left we see the input image and on the right the encrypted picture produced by the algorithm.

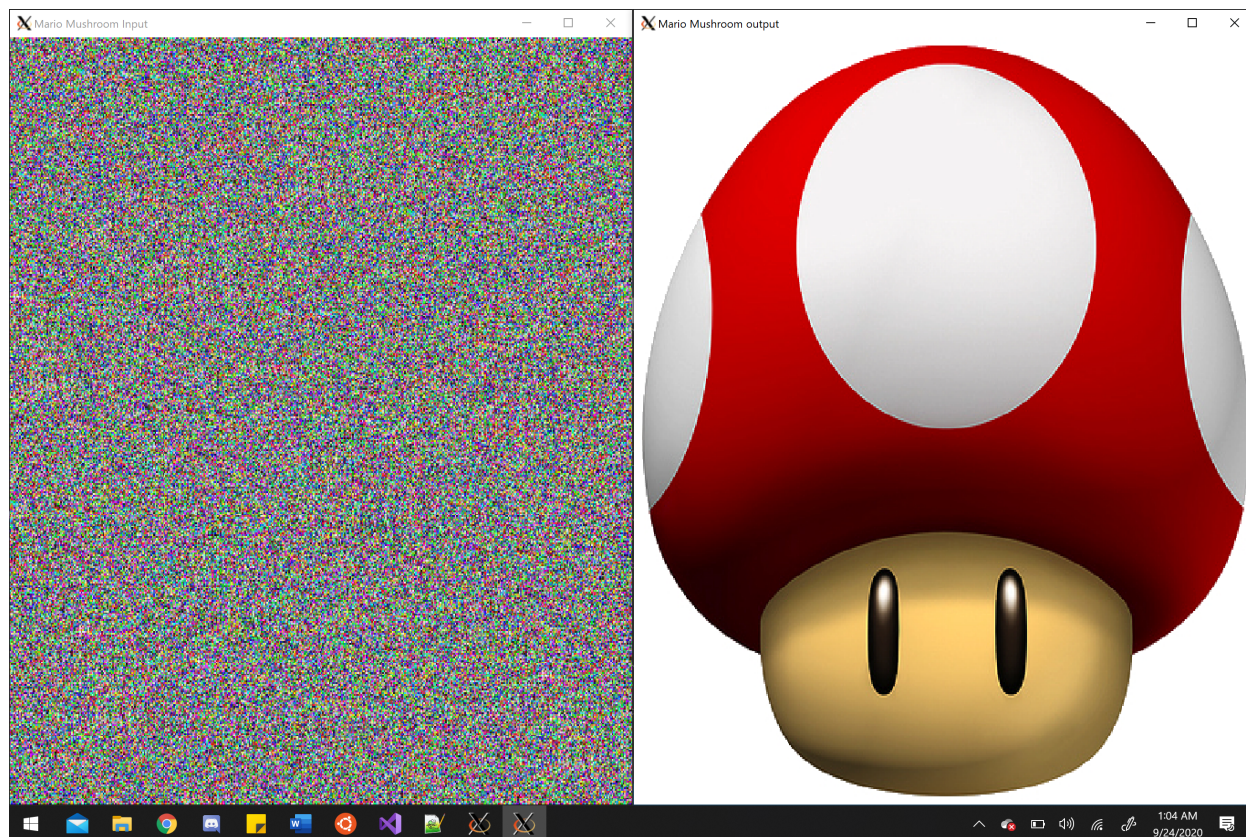
Figure 4: Encryption of Mario Mushroom



### 3.9 decode.png

On the left we have the encrypted image and on the right the decrypted image produced by reversing the algorithm.

Figure 5: Decryption of Mario Mushroom



Overall, I mostly learned about sprite manipulation and how you can use one part of one project for a different kind of project in another. Breaking down a big project helped me understand what I need to accomplish first in order to move on to other parts of the projects.

## 4 Ps2a - N-Body Simulation Part A

### 4.1 Overview

This project consisted of parts A and B. In this part of this assignment, we had to create a program that could read in information about the positions of celestial bodies in a universe and draw them using the provided sprites and scaling information so that the universe could be visualized. As a result, the program will load and display a static universe. When reading in the universe, the user inputs a file that contains the information for a particular universe. The first value is an integer  $N$  which represents the number of particles. The second value is a real number  $R$  which represents the radius of the universe, used to determine the scaling of the drawing window. Finally, there are  $N$  rows, and each row contains 6 values. The first two values are the  $x$ - and  $y$ -coordinates of the initial position; the next pair of values are the  $x$ - and  $y$ -components of the initial velocity; the fifth value is the mass; the last value is a String that is the name of an



image file used to display the particle. For example, we will be inputting the planets.txt file: For

Figure 6: Example of Planets.txt File

...xpos...	...ypos...	...xvel...	...yvel...	...mass...	filename
1.4960e+11	0.0000e+00	0.0000e+00	2.9800e+04	5.9740e+24	earth.gif
2.2790e+11	0.0000e+00	0.0000e+00	2.4100e+04	6.4190e+23	mars.gif
5.7900e+10	0.0000e+00	0.0000e+00	4.7900e+04	3.3020e+23	mercury.gif
0.0000e+00	0.0000e+00	0.0000e+00	0.0000e+00	1.9890e+30	sun.gif
1.0820e+11	0.0000e+00	0.0000e+00	3.5000e+04	4.8690e+24	venus.gif

part A of this assignment, we had to implement a Universe class which will contain all celestial bodies, and a CelestialBody class representing the celestial bodies. The class Universe should create celestial bodies. We also had to override the input stream operator », and use it to load parameter data into an object while using smart pointers.

## 4.2 Key Concepts

In this assignment we had to use smart pointers to manage objects that had to be passed around and shared. Learning to read input from command line was also a reoccurring task in these assignments.

## 4.3 Learnings and Accomplishments

I had not used smart pointers before so I had to learn how to use them. I think smart pointers are much safer and more efficiently used compared to regular pointers so I am glad I learned them and will utilize them in my future programs.

## 4.4 Makefile

The makefile was the same format, with the additional libraries graphics, window, and system used from the SFML library.

```
1 CC = g++
2 CFLAGS = -g -w -Wall -Werror -ansi -pedantic
3 LDLIBS = -lsfml-graphics -lsfml-window -lsfml-system
4
5 NBody: NBody.o main.o
6     $(CC) $(CFLAGS) -o NBody NBody.o main.o $(LDLIBS)
7 all: NBody
8 NBody.o: NBody.cpp NBody.h
9     $(CC) $(CFLAGS) -c NBody.cpp
10 main.o: main.cpp
11     $(CC) $(CFLAGS) -c main.cpp
12
13 clean:
14     rm *.o NBody
```

## 4.5 main.cpp

In the main file, I had created two variables and read them into command line by using cin. Afterwards, I created a Universe Object and then passed in the radius, the window size, the number of celestial bodies, and the cin input. I then made a window and while that window was open, the event would run and display the Universe object. If the window was closed, the window would close as well.

```
1 #include <SFML/Graphics.hpp>
2 #include <SFML/Window.hpp>
3 #include <SFML/System.hpp>
4 #include <iostream>
5 #include <vector>
6 #include "NBody.h"
7
8
9 using namespace std;
10 int main()
11 {
12
13     int num_bodies;
14     double radius;
15     std::cin >> num_bodies;
16     std::cin >> radius;
17
18
19     Universe cb(radius, 500, num_bodies, std::cin);
20     //event window
21     sf::RenderWindow window(sf::VideoMode(500, 500), "Solar System")
22     ;
23     while (window.isOpen())
24     {
25         sf::Event event;
26         while (window.pollEvent(event))
27         {
28             if (event.type == sf::Event::Closed)
29             {
30                 window.close();
31             }
32         }
33         window.clear();
34         window.draw(cb);
35         window.display();
36     }
37     return 0;
38 }
```

## 4.6 NBody.h

In the NBody header file, I created two different classes: Universe and CelestialBody. I made two classes since there were similar functions that had to be used for other celestial bodies. In the CelestialBody class, I had created a bunch of private variables for x and y like their positions and their velocity. I had also made a mass, radius, and string for the filename, a sprite object, and a texture object. In my public functions, I added accessor and mutator functions such as get\_posx() and set\_position() in order to use my private variables. In my Universe class, I made a vector of CelestialBody pointers so I could allocate space for them if needed to contain data.

```
1 #ifndef NBODY_H
2 #define NBODY_H
3 #include <SFML/Graphics.hpp>
4 #include <string>
5 #include <vector>
6 #include <iostream>
7
8
9 class CelestialBody :public sf::Drawable {
10 private:
11     double windowSize;
12     double xpos;
13     double ypos;
14     double xvel;
15     double yvel;
16     double mass;
17     double radius;
18     std::string filename;
19     sf::Sprite sprite;
20     sf::Texture texture;
21
22 public:
23     //constructors
24     CelestialBody();
25     CelestialBody(double x_pos, double y_pos, double x_vel, double
        y_vel, double m, std::string name, double radius, double
        windowSize);
26     ~CelestialBody();
27     //istream
28     friend std::istream& operator >>(std::istream& input,
        CelestialBody& ci);
29     //draw function
30     virtual void draw(sf::RenderTarget &target, sf::RenderStates
        states) const;
31     //accessor functions
32     double get_posx();
33     double get_posy();
34     //mutators
35     void set_radius(double radius);
```

```

36     void set_window(double size);
37     void set_position();
38     // sf::Sprite& get_sprite() { return sprite; };
39 };
40
41 class Universe : public sf::Drawable {
42 public:
43     Universe();
44     Universe(double radius, int window, int num_of_planets, std::
        istream &in); //creates/ allocates space for a celestialbody,
        completely empty
45     virtual void draw(sf::RenderTarget& target, sf::RenderStates
        states) const;
46
47 private:
48     double r;
49     int windowSize;
50     std::vector<CelestialBody*> planets; //allocating space for them
51     int numberOfPlanets;
52
53 };
54 #endif //NBody_H

```

## 4.7 NBody.cpp

In the NBody.cpp file, I started creating the function definitions for each class. We had to initialize each variable and then for the constructor, we would push back a new CelestialBody pointer into the vector. Then we would dereference to set the radius and window. In the draw function, we would iterate through the number of planets draw each of the planets at that index. In the CelestialBody class, we would also initialize our variables, and then set the position by saving the calculations into the xpos and ypos variables so they would be drawn to screen with the original coordinates. I also overloaded the instream operator to take in input for the necessary coordinates and loaded the images of each celestial body, set the texture, and the position. I got confused over the SFML library function set position and the same one I wrote which had similar names. The rest were single lined functions that were either accessor or mutator functions to access the variables.

```

1 #include "NBody.h"
2
3 Universe::Universe() {
4     r = 0;
5     windowSize = 0;
6 }
7
8 Universe::Universe(double radius, int window, int num_of_planets,
    std::istream& in) {
9     //set radius, windows
10    r = radius;

```

```

11     windowSize = window;
12     numberOfPlanets = num_of_planets;
13
14     //initialize planets
15     for (int i = 0; i < num_of_planets; i++) {
16         planets.push_back(new CelestialBody()); //allocate space for
            a CelestialBody, and pushes allocated pointer into
            vector
17         planets[i]->set_radius(r); //planets[i] is a pointer to the
            object
18         planets[i]->set_window(window);
19         in >> *planets[i];
20     }
21
22 }
23
24 void Universe::draw(sf::RenderTarget& target, sf::RenderStates
    states) const {
25     for (int i = 0; i < numberOfPlanets; i++) {
26         target.draw(*planets.at(i), states); //draw each of the
            planets
27     }
28
29 }
30
31
32 CelestialBody::CelestialBody() {
33     windowSize = 0;
34     xpos = 0;
35     ypos = 0;
36     xvel = 0;
37     yvel = 0;
38     mass = 0;
39     radius = 0;
40     filename = "";
41
42 }
43
44 CelestialBody::CelestialBody(double x_pos, double y_pos, double
    x_vel, double y_vel, double m, std::string name, double rad,
    double window_size) {
45     xpos = x_pos;
46     ypos = y_pos;
47     xvel = x_vel;
48     yvel = y_vel;
49     mass = m;
50     radius = rad;
51     windowSize = window_size;

```



```

52     filename = name;
53
54
55
56     //fit universe onto screen
57     xpos = (windowSize / 2) * (xpos / radius) + (windowSize / 2);
58     ypos = (windowSize / 2) * (ypos / radius) + (windowSize / 2);
59
60
61
62     texture.loadFromFile(filename);    //Load Texture from image
63
64
65     sprite.setTexture(texture);
66     sprite.setPosition(xpos, ypos);
67 }
68
69 std::istream& operator >>(std::istream& in, CelestialBody& ci) {
70     in >> ci.xpos >> ci.ypos >> ci.xvel
71         >> ci.yvel >> ci.mass >> ci.filename;
72
73     ci.xpos = (ci.windowSize / 2) * (ci.xpos / ci.radius) + (ci.
        windowSize / 2);
74     ci.ypos = (ci.windowSize / 2) * (ci.ypos / ci.radius) + (ci.
        windowSize / 2);
75
76     ci.texture.loadFromFile(ci.filename);    //Load Texture from image
77
78
79     ci.sprite.setTexture(ci.texture);
80     ci.sprite.setPosition(ci.xpos, ci.ypos);
81
82
83     return in;
84 }
85
86 void CelestialBody::draw(sf::RenderTarget &target, sf::RenderStates
    states) const {
87     target.draw(sprite, states);
88 }
89
90 void CelestialBody::set_position() {
91
92     double result = (windowSize / 2) / radius;
93     double rxpos = xpos * result + (windowSize / 2);
94     double rypos = ypos * result + (windowSize / 2);
95     sprite.setPosition(sf::Vector2f(rxpos, rypos));
96 }

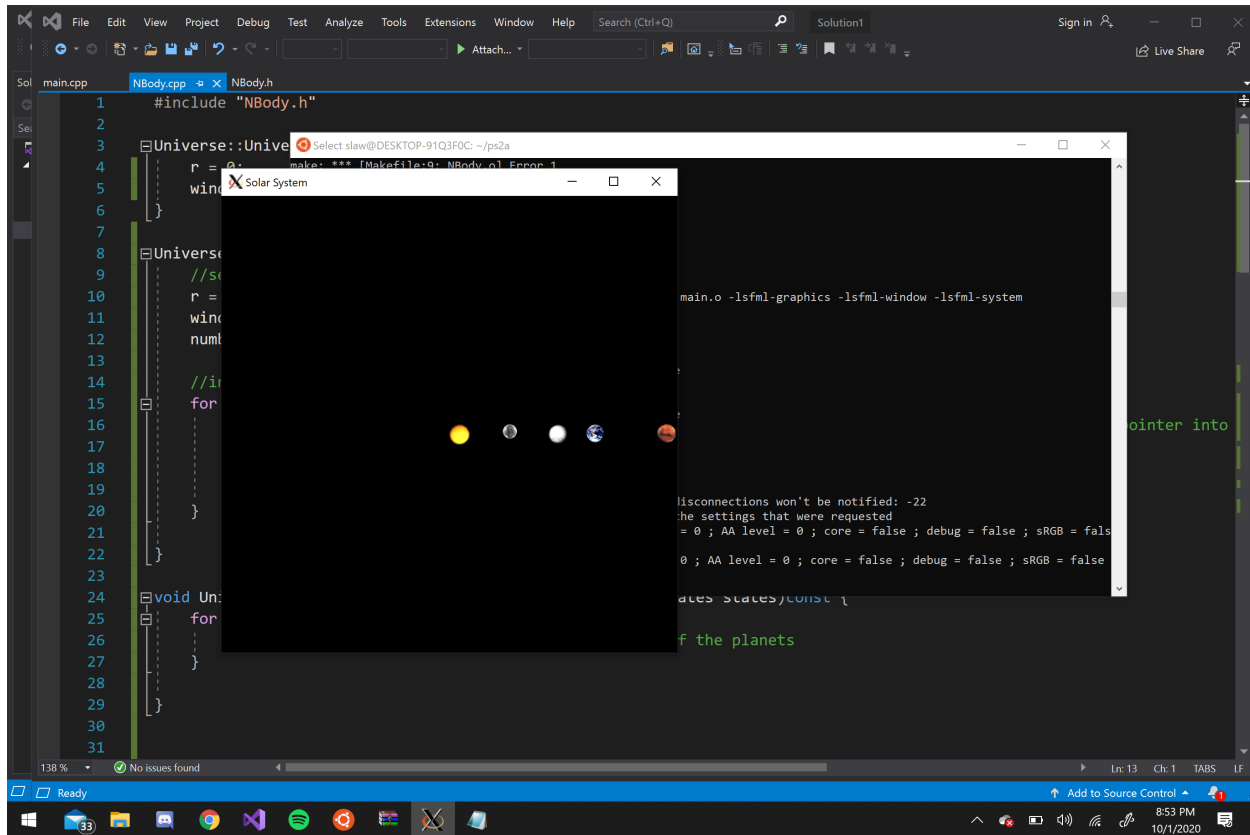
```

```
97
98 CelestialBody::~~CelestialBody() {
99
100 }
101
102 double CelestialBody::get_posx() {
103     return xpos;
104 }
105
106 double CelestialBody::get_posy() {
107     return ypos;
108 }
109
110 void CelestialBody::set_radius(double r) {
111     radius = r;
112 }
113
114 void CelestialBody::set_window(double size) {
115     windowSize = size;
116 }
```

## 4.8 screenshot.png

When given the coordinates, a window will display our static universe with the sun and the four innermost planets around it.

Figure 7: Screenshot of a Static Solar System from Mercury to Mars



## 5 Ps2b - N-Body Simulation Part B

### 5.1 Overview

In this assignment we had to create a physics simulation to bring our universes from part A in motion. We wrote a program that accepts the same parameters as the one specified, and reads the universe file from stdin. We had to name our executable NBody, so we would run it with: We

Figure 8: Example of Command Line

```
./NBody 157788000.0 25000.0 < planets.txt
```

used the leapfrog finite difference approximation scheme to numerically integrate the above equations: this is the basis for most astrophysical simulations of gravitational systems. In the leapfrog scheme, we discretize time, and update the time variable  $t$  in increments of the time quantum  $\Delta t$  (measured in seconds). We maintain the position  $(px, py)$  and velocity  $(vx, vy)$  of each particle at each time step. The steps below illustrate how to evolve the positions and velocities of the particles.

### 5.2 Key Concepts

We used a leapfrog approximation with small time steps to approximate solutions to the differential equations that govern motion. We also used the SFML audio in order to create sound for the universe.

### 5.3 Learnings and Accomplishments

### 5.4 Makefile

The makefile was the same format, with the additional libraries graphics, window, and system used from the SFML library with audio included this time.

```
1 CC = g++
2 CFLAGS = -g -w -Wall -Werror -ansi -pedantic -std=c++11
3 LDLIBS = -lsfml-graphics -lsfml-window -lsfml-audio -lsfml-system
4
5 NBody: NBody.o main.o
6     $(CC) $(CFLAGS) -o NBody NBody.o main.o $(LDLIBS)
7 all: NBody
8 NBody.o: NBody.cpp NBody.h
9     $(CC) $(CFLAGS) -c NBody.cpp
10 main.o: main.cpp
11     $(CC) $(CFLAGS) -c main.cpp
12
13 clean:
14     rm *.o NBody
```

## 5.5 main.cpp

In my main.cpp, I did not make drastic changes to the overall file, but I also added in a soundbuffer and loaded a sound from file in order to play it when the window was open. I also created two variables T and delta T to represent the time. I made a loop while time passed is less than T, then call the step function and add that time to T.

```
1 // Copyright 2020 Stephanie La
2
3 #include <SFML/Graphics.hpp>
4 #include <SFML/Window.hpp>
5 #include <SFML/System.hpp>
6 #include <SFML/Audio.hpp>
7 #include <iostream>
8 #include <cstdlib>
9 #include <vector>
10 #include <string> // NOLINT
11 #include "NBody.h"
12
13
14 // compare with other bodies
15 // apply force x and y
16 // apply step function
17
18 int main(int argc, char* argv[]) {
19     if (argc != 3) {
20         std::cout << "Not enough arguments" << std::endl;
21         return -1;
22     }
23
24     int num_bodies;
25     double radius;
26     std::string filename;
27     double time_passed = 0;
28     filename = argv[0];
29     // max time
30     double T = strtod(argv[1], NULL);
31     double deltaT = strtod(argv[2], NULL);
32
33     std::cin >> num_bodies;
34     std::cin >> radius;
35
36
37     Universe cb(radius, 500, num_bodies, std::cin);
38     // event window
39
40     sf::RenderWindow window(sf::VideoMode(600, 600), "Solar System");
41     // makes sound
42     sf::SoundBuffer buffer;
```

```

43  if (!buffer.loadFromFile("2001.wav")) {
44  return -1;
45  }
46  sf::Sound sound;
47  sound.setBuffer(buffer);
48  sound.play();
49
50  while (window.isOpen()) {
51  sf::Event event;
52  while (window.pollEvent(event)) {
53  if (event.type == sf::Event::Closed) {
54  window.close();
55  }
56  }
57      window.clear();
58      if (time_passed < T) {
59          cb.step(deltaT);
60          time_passed += deltaT;
61      }
62      window.draw(cb);
63      window.display();
64  }
65  // close the solar system window to see the input information
66  cb.printInfo();
67  return 0;
68 }

```

## 5.6 NBody.h

In my header, file, I did not also make any drastic changes from part A, but I added a couple more accessor and mutator files for each variable to my public members of the CelestialBody and Universe class.

```

1 // Copyright 2020 Stephanie La
2
3 #ifndef NBODY_H // NOLINT
4 #define NBODY_H // NOLINT
5 #include <SFML/Graphics.hpp>
6 #include <string>
7 #include <vector>
8 #include <iostream>
9 #include <memory>
10
11
12 class CelestialBody :public sf::Drawable {
13 private:
14     double windowSize;
15     double xpos;

```

```

16     double ypos;
17     double xvel;
18     double yvel;
19     double mass;
20     double radius;
21     double display_x;
22     double display_y;
23     std::string filename;
24     sf::Sprite sprite;
25     sf::Texture texture;
26
27 public:
28     // constructors
29     CelestialBody();
30     CelestialBody(double x_pos, double y_pos, double x_vel,
31     double y_vel, double m, std::string name, double radius, double
        windowSize);
32     ~CelestialBody();
33     // istream
34     friend std::istream& operator >>(std::istream& input,
        CelestialBody& ci);
35     // ostream
36     // friend std::ostream& operator <<(std::ostream& out,
        CelestialBody& co);
37     // draw function
38     virtual void draw(sf::RenderTarget& target, // NOLINT
39     sf::RenderStates states) const; // NOLINT
40     // accessor functions
41     double get_posx();
42     double get_posy();
43     double get_velx();
44     double get_vely();
45     double get_mass();
46     std::string get_filename();
47     // mutators
48     void set_x_y_pos(double x_input, double y_input);
49     void set_velx(double vx);
50     void set_vely(double vy);
51     void set_radius(double radius);
52     void set_window(double size);
53     void set_position();
54 };
55
56 class Universe : public sf::Drawable {
57 public:
58     Universe();
59     Universe(double radius, int window,
60     // creates allocates space for a celestialbody, completely empty

```

```

61     int num_of_planets, std::istream& in);
62     virtual void draw(sf::RenderTarget& target, // NOLINT
63         sf::RenderStates states) const; // NOLINT
64     // ostream
65     friend std::ostream& operator <<(std::ostream& out, const
        Universe& co);
66     void step(double seconds);
67     void printInfo();
68     // accessor functions
69     double get_r();
70     int get_numPlanets();
71 private:
72     double r;
73     int windowSize;
74     // std::vector<CelestialBody*> planets;
75     // allocating space for them
76     std::vector<std::unique_ptr<CelestialBody>> planets;
77     int numberOfPlanets;
78 };
79 #endif // NBODY_H // NOLINT

```

## 5.7 NBody.cpp

In my NBody.cpp file, I added in the step() function from the Universe class where I have to implement the calculations of each celestial bodies. The order of how each line was placed, because you could not call variables that were not declared yet, you would get a garbage value that would cause the universe's rotation to go off. For example, you would have to save the calculations for a radius and then use it in another equation, but if you use the radius first without defining it, you will have a garbage value assigned to it, and the entire calculation would be wrong. When printing the information and coordinates of each planet, I used a function to do that instead of implementing it in my main function. I used a loop to print each piece of information out for efficiency.

```

1 // Copyright 2020 Stephanie La
2
3 // #include "/home/slaw/ps2b/NBody.h" // NOLINT
4
5 #include "NBody.h" // NO LINT
6 #include <cmath>
7 #include <memory>
8 #include <utility> // NO LINT
9 #include <string>
10
11 Universe::Universe() {
12     r = 0;
13     windowSize = 0;
14 }
15

```



```

16 Universe::Universe(double radius, int window, int num_of_planets,
17     std::istream& in) {
18     // set radius, windows
19     r = radius;
20     windowSize = window;
21     numberOfPlanets = num_of_planets;
22     // initialize planets
23     for (int i = 0; i < num_of_planets; i++) {
24         // cant just convert a CelestialBody* into a smart pointer
25         std::unique_ptr<CelestialBody>ptr(new CelestialBody());
26         // assign a new CelestialBody to that pointer
27         CelestialBody();
28         planets.push_back(std::move(ptr));
29         // planets.push_back(new CelestialBody());
30         // allocate space for a CelestialBody, and pushes
31         // allocated pointer into vector
32         // planets[i] is a pointer to the object
33         planets[i]->set_radius(r);
34         planets[i]->set_window(window);
35         in >> *planets[i];
36     }
37 }
38
39 void Universe::draw(sf::RenderTarget& target, sf::RenderStates
    states)const {
40     for (int i = 0; i < numberOfPlanets; i++) {
41         // draw each of the planets
42         target.draw(*planets.at(i), states);
43     }
44 }
45
46 // fix here
47 void Universe::step(double seconds) {
48     // for each planet, implement net forces
49     for (int i = 0; i < numberOfPlanets; i++) {
50         // all horizontal forces on a body
51         double net_fx = 0;
52         double net_fy = 0;
53         // for each planet, calculate total forces
54         // outside force
55         for (int k = 0; k < numberOfPlanets; k++) {
56             if (k != i) {
57                 // gravitational constant
58                 double G = 6.67e-11;
59                 // compiler reads top down
60                 double deltaX = planets[k]->get_posx() - planets[i]->get_posx();
61                 double deltaY = planets[k]->get_posy() - planets[i]->get_posy();
62                 // double Force = (6.67e-11 * planets[k]->get_mass())

```

```

63 // * planets[i]->get_mass()) / pow(r, 2);
64 double r = sqrt(pow(deltaX, 2) + pow(deltaY, 2));
65 double Force = (G * planets[k]->get_mass() *
66 planets[i]->get_mass()) / pow(r, 2);
67 double forceX = Force * (deltaX / r);
68 double forceY = Force * (deltaY / r);
69 // adding force to total net force acting on body
70 net_fy += forceY;
71 net_fx += forceX;
72 }
73 }
74 double accX = net_fx / planets[i]->get_mass();
75 double accY = net_fy / planets[i]->get_mass();
76 double velX = planets[i]->get_velx() + seconds * accX;
77 double velY = planets[i]->get_vely() + seconds * accY;
78 planets[i]->set_velx(velX);
79 planets[i]->set_vely(velY);
80 double newX = (planets[i]->get_posx()) + velX * seconds;
81 double newY = (planets[i]->get_posy()) + velY * seconds;
82 planets[i]->set_x_y_pos(newX, newY);
83 }
84 }
85
86 // prints info out to screen
87 void Universe::printInfo() {
88     std::cout << numberOfPlanets << std::endl;
89     std::cout << r << std::endl;
90     for (int i = 0; i < numberOfPlanets; i++) {
91         std::cout << planets[i]->get_posx() << " " << planets[i]->
            get_posy()
92         << " " << planets[i]->get_velx() << " " << planets[i]->get_vely
            () << " "
93         << planets[i]->get_mass() << " " << planets[i]->get_filename() <<
            std::endl;
94     }
95 }
96
97 double Universe::get_r() {
98     return r;
99 }
100
101 int Universe::get_numPlanets() {
102     return numberOfPlanets;
103 }
104
105 // find force
106 // step used on one body
107 CelestialBody::CelestialBody() {

```

```

108     windowSize = 0;
109     xpos = 0;
110     ypos = 0;
111     xvel = 0;
112     yvel = 0;
113     mass = 0;
114     radius = 0;
115     filename = "";
116 }
117
118 CelestialBody::CelestialBody(double x_pos, double y_pos,
119     double x_vel, double y_vel, double m, std::string name,
120     double rad, double window_size) {
121     xpos = x_pos;
122     ypos = y_pos;
123     xvel = x_vel;
124     yvel = y_vel;
125     mass = m;
126     radius = rad;
127     windowSize = window_size;
128     filename = name;
129     // fit universe onto screen
130     /* xpos = (windowSize / 2) * (xpos / radius) + (windowSize / 2);
131     ypos = (windowSize / 2) * (ypos / radius) + (windowSize / 2); */
132     double rxpos = (windowSize / 2) * (xpos / radius) + (windowSize /
133         2);
134     double rypos = (windowSize / 2) * (ypos / radius) + (windowSize /
135         2);
136     // Load Texture from image
137     texture.loadFromFile(filename);
138     sprite.setTexture(texture);
139     sprite.setPosition(rxpos, rypos);
140 }
141
142 std::istream& operator >>(std::istream& in, CelestialBody& ci) {
143     in >> ci.xpos >> ci.ypos >> ci.xvel
144     >> ci.yvel >> ci.mass >> ci.filename;
145     /* ci.xpos = (ci.windowSize / 2) * (ci.xpos / ci.radius) + (ci.
146         windowSize / 2);
147     ci.ypos = (ci.windowSize / 2) * (ci.ypos / ci.radius)
148     + (ci.windowSize / 2); */
149     double rxpos = (ci.windowSize / 2) * (ci.xpos / ci.radius)
150     + (ci.windowSize / 2);
151     double rypos = (ci.windowSize / 2) * (ci.ypos / ci.radius)
152     + (ci.windowSize / 2);
153     // Load Texture from image
154     ci.texture.loadFromFile(ci.filename);
155     ci.sprite.setTexture(ci.texture);

```

```

153     ci.sprite.setPosition(rxpos, rypos);
154     return in;
155 }
156
157 void CelestialBody::draw(sf::RenderTarget& target,
158     sf::RenderStates states) const {
159     target.draw(sprite, states);
160 }
161
162 // void CelestialBody::set_position() {
163 //     double result = (windowSize / 2) / radius;
164 //     double rxpos = xpos * result + (windowSize / 2);
165 //     double rypos = ypos * result + (windowSize / 2);
166 //     sprite.setPosition(sf::Vector2f(rxpos, rypos));
167 // }
168
169 CelestialBody::~CelestialBody() {
170 }
171
172 double CelestialBody::get_posx() {
173     return xpos;
174 }
175
176 double CelestialBody::get_posy() {
177     return ypos;
178 }
179
180 double CelestialBody::get_velx() {
181     return xvel;
182 }
183
184 double CelestialBody::get_vely() {
185     return yvel;
186 }
187
188 double CelestialBody::get_mass() {
189     return mass;
190 }
191
192 std::string CelestialBody::get_filename() {
193     return filename;
194 }
195
196 void CelestialBody::set_radius(double r) {
197     radius = r;
198 }
199
200 void CelestialBody::set_window(double size) {

```

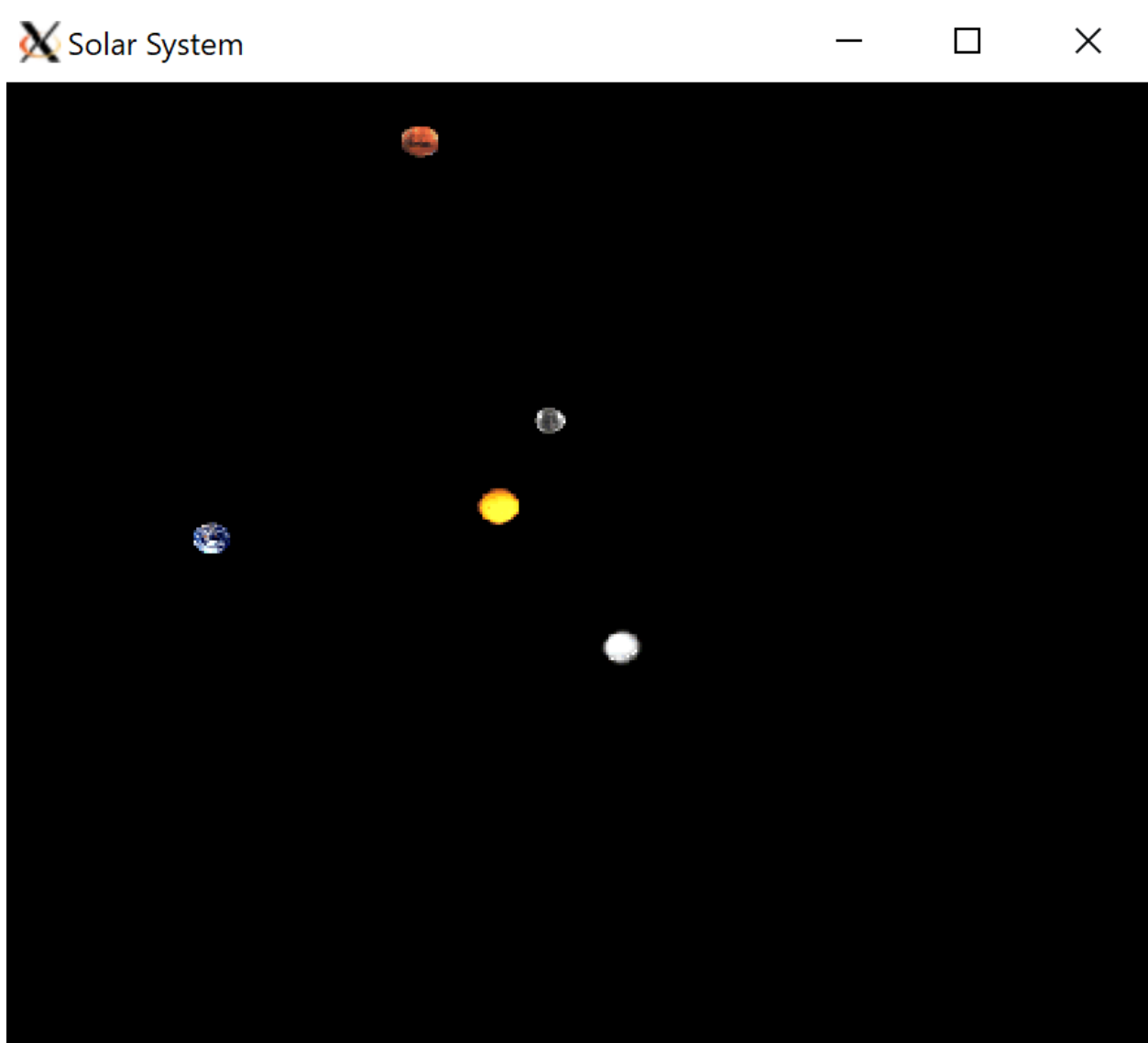
```

201     windowSize = size;
202 }
203
204 void CelestialBody::set_velx(double vx) {
205     xvel = vx;
206 }
207 void CelestialBody::set_vely(double vy) {
208     yvel = vy;
209 }
210
211 void CelestialBody::set_x_y_pos(double x_input, double y_input) {
212     xpos = x_input;
213     ypos = y_input;
214     double rxpos = (windowSize / 2) * (xpos / radius) + (windowSize /
215         2);
216     double rypos = (windowSize / 2) * (-ypos / radius) + (windowSize /
217         2);
218     sprite.setPosition(sf::Vector2f(rxpos, rypos));
219 }

```

## 5.8 Screenshot of Planets

Figure 9: Example of Command Line



## 6 Ps3a - Synthesizing a Plucked String Sound Part A

### 6.1 Overview

In this assignment we had to create a CircularBuffer class which acted as a data structure with a fixed maximum capacity. We had to write unit tests to ensure our code was working properly. We also used the boost library again to test if the CircleBuffer worked because we didn't have anything to test it with yet. I also uses runtime errors in enqueue when the CircleBuffer is full and something try's to call enqueue and when its empty and something try's to call dequeue.

## 6.2 Key Concepts

In this assignment we were introduced to C++ exceptions and a big focus of the testing was on ensuring that the correct exceptions were thrown and or not thrown for a particular situation. We also learned about cpplint a tool for checking the style of C++ code.

## 6.3 Learnings and Accomplishments

I had not heard of a CircularBuffer data structure before so I had to do some research to learn about how they work and how to implement one. I chose to use a fixed size array and two ints, one holding the start point of the data and another holding how many elements are in the buffer. I also had to learn about how to use cpplint so that I could get my code to pass its tests. The process was a bit difficult, but seeing the message that I had 0 errors was very satisfying that I was able to implement this structure correctly.

## 6.4 Makefile

The makefile was the same format, but we only used the SFML audio library for this program. I also included the boost library framework to test the functions.

```
1 CC = g++
2 CFLAGS = -g -w -Wall -Werror -ansi -pedantic -std=c++11
3
4 ps3a: CircularBuffer.o test.o
5     $(CC) $(CFLAGS) -o ps3a CircularBuffer.o test.o -
        lboost_unit_test_framework
6 all: ps3a
7 CircularBuffer.o: CircularBuffer.cpp CircularBuffer.h
8     $(CC) $(CFLAGS) -c CircularBuffer.cpp
9 test.o: test.cpp
10    $(CC) $(CFLAGS) -c test.cpp
11 clean:
12    rm *.o ps3a
```

## 6.5 test.cpp

In my test.cpp file, I tested if each function was working and threw invalid arguments to test if the function was working correctly. As a result, it worked normally with enqueue and dequeue functions. I also implemented a lambda function into the program to automatically calculate one of the equations.

```
1 // Copyright 2020 Stephanie La
2 #define BOOST_TEST_DYN_LINK
3 #define BOOST_TEST_MODULE Main
4
5 #include <stdint.h>
6 #include <boost/test/unit_test.hpp>
7 #include <iostream>
8 #include <string>
9 #include <exception>
```

```

10 #include <stdexcept>
11 #include "CircularBuffer.h"
12
13 // comment out one or another to test
14
15 BOOST_AUTO_TEST_CASE(build_constructor) {
16     // comment out one or another to test
17     BOOST_REQUIRE_NO_THROW(CircularBuffer(20));
18
19     BOOST_REQUIRE_THROW(CircularBuffer(0), std::invalid_argument);
20 }
21
22 BOOST_AUTO_TEST_CASE(test_enqueue_and_dequeue) {
23     // make object
24     CircularBuffer cb(3);
25
26     // lambda
27     auto check_dq = [&](int16_t x) { return cb.dequeue() == x; };
28
29     // enqueue
30     cb.enqueue(2);
31     cb.enqueue(3);
32     cb.enqueue(4);
33     BOOST_REQUIRE(cb.peek() == 2);
34     BOOST_REQUIRE_THROW(cb.enqueue(5), std::runtime_error);
35     // dequeue
36
37     BOOST_REQUIRE(check_dq(2));
38     BOOST_REQUIRE(check_dq(3));
39     BOOST_REQUIRE(check_dq(4));
40
41
42     /* BOOST_REQUIRE(cb.dequeue() == 2);
43     BOOST_REQUIRE(cb.dequeue() == 3);
44     BOOST_REQUIRE(cb.dequeue() == 4);*/
45
46     BOOST_REQUIRE_THROW(cb.dequeue(), std::runtime_error);
47 }
48
49 BOOST_AUTO_TEST_CASE(test_peek) {
50     CircularBuffer cb(3);
51
52     cb.enqueue(10);
53     cb.enqueue(50);
54     cb.enqueue(100);
55     BOOST_REQUIRE(cb.peek() == 10);
56     BOOST_REQUIRE(cb.dequeue() == 10);
57     BOOST_REQUIRE(cb.peek() == 50);

```



## 6.6 CircularBuffer.h

In my header file, I decided to write CircularBuffer in a different way, I only had three integer variables, the capacity, front, and the bufferSize. I figured that I could figure out the back of the queue with the front variable. I had also used a unique pointer of int16\_t because it would not resize efficiently as a regular vector.

```

1 // Copyright 2020 Stephanie La
2
3 // #ifndef CIRCULARBUFFER_H // NOLINT
4 // #define CIRCULARBUFFER_H // NOLINT
5 #ifndef _HOME_SLAWS_PS3A_CIRCULARBUFFER_H_
6 #define _HOME_SLAWS_PS3A_CIRCULARBUFFER_H_
7 #include <stdint.h>
8 #include <iostream>
9 #include <vector>
10 #include <memory>
11
12 class CircularBuffer {
13 public:
14 // create an empty ring buffer, with given max
15 explicit CircularBuffer(int capacity);
16 // capacity
17 // return number of items currently in the buffer
18 int size();
19 // is the buffer empty (size equals zero)?
20 bool isEmpty();
21 // is the buffer full (size equals capacity)?
22 bool isFull();
23 // add item x to the end
24 void enqueue(int16_t x);
25 // delete and return item from the front
26 int16_t dequeue();
27 // return (but do not delete) item from the front
28 int16_t peek();
29
30 private:
31 std::unique_ptr<int16_t[]>buffer;
32 int bufferSize;
33 int bufferCapacity;
34 int front;
35 };
36 #endif // _HOME_SLAWS_PS3A_CIRCULARBUFFER_H_

```

## 6.7 CircularBuffer.cpp

In this cpp file, I implemented calculations and made a temporary variable called back to keep track of the calculation to mod the buffer capacity and then saved the result and incremented the size since it was a fixed array. Many of the functions wanted to check if it was full or not and throw an error if it was not full or empty.

```
1 // Copyright 2020 Stephanie La
2
3 // #include "CircularBuffer.h"
4 #include "/home/slaw/ps3a/CircularBuffer.h"
5 #include <stdint.h>
6 #include <iostream>
7 #include <stdexcept>
8 #include <vector>
9 #include <memory>
10
11 CircularBuffer::CircularBuffer(int capacity) {
12     if (capacity < 1) {
13         throw std::invalid_argument(
14             "CB constructor: capacity must be greater than zero");
15     }
16     bufferCapacity = capacity;
17     bufferSize = 0;
18     front = 0;
19     buffer = std::unique_ptr<int16_t[]>(new int16_t[capacity]);
20 }
21
22 int CircularBuffer::size() {
23     return bufferSize;
24 }
25
26 bool CircularBuffer::isEmpty() {
27     return bufferSize == 0;
28 }
29
30 bool CircularBuffer::isFull() {
31     return bufferSize == bufferCapacity;
32 }
33
34 void CircularBuffer::enqueue(int16_t x) {
35     if (isFull()) {
36         throw std::runtime_error("enqueue: can't enqueue to a full
37             ring.");
38     }
39     // back declaration
40     int back = (front + bufferSize) % bufferCapacity;
41     buffer[back] = x;
42     bufferSize++;
```

```

42 }
43
44 int16_t CircularBuffer::dequeue() {
45     if (isEmpty()) {
46         throw std::runtime_error("Buffer is empty");
47     }
48     // save
49     int16_t temp = buffer[front];
50     front = (front + 1) % bufferCapacity;
51     bufferSize--;
52     return temp;
53 }
54
55 int16_t CircularBuffer::peek() {
56     if (isEmpty()) {
57         throw std::runtime_error("Buffer is empty");
58     }
59     return buffer[front];
60     // front is at index front
61     // holds the index where the data begins
62     // return buffer.at(front); // peek "front" of vector buff also
        valid
63 }

```

## 7 Ps3b - Synthesizing a Plucked String Sound Part B

### 7.1 Overview

In this assignment we had to synthesize audio sounding like a guitar string using our CircleBuffer. A CircleBuffer is just a queue that hold int16 t's. A CircleBuffer uses the API for a queue.

### 7.2 Key Concepts

In this assignment we were introduced to C++ exceptions and a big focus of the testing was on ensuring that the correct exceptions were thrown and or not thrown for a particular situation. We also learned about cplint a tool for checking the style of C++ and how to practice good syntax.

### 7.3 Learnings and Accomplishments

I think the most difficult part was learning how to work and implement a fixed data structure, understand how to make multiple vectors of samples and passing them into each other to be able to play the sound from samples to vector samples.

### 7.4 Makefile

The makefile was the same format, but we had to include a few extra files and just used the SFML audio library to compile the program.

```

1 CC = g++
2 CFLAGS = -g -w -Wall -Werror -ansi -pedantic -std=c++11
3
4 KSGuitarSim: CircleBuffer.o StringSound.o KSGuitarSim.o
5     $(CC) $(CFLAGS) -o KSGuitarSim CircleBuffer.o StringSound.o
        KSGuitarSim.o -lsfml-graphics -lsfml-window -lsfml-audio -
        lsfml-system
6 all: KSGuitarSim
7 CircularBuffer.o: CircleBuffer.cpp CircleBuffer.h
8     $(CC) $(CFLAGS) -c CircleBuffer.cpp
9 StringSound.o: StringSound.cpp StringSound.h
10    $(CC) $(CFLAGS) -c StringSound.cpp
11 KSGuitarSim.o: KSGuitarSim.cpp
12    $(CC) $(CFLAGS) -c KSGuitarSim.cpp
13 clean:
14     rm *.o KSGuitarSim

```

## 7.5 KSGuitarSim.cpp

```

1 /*
2  Copyright 2015 Fred Martin,
3  Y. Rykalova, 2020
4 */
5
6 #include <SFML/Graphics.hpp>
7 #include <SFML/System.hpp>
8 #include <SFML/Audio.hpp>
9 #include <SFML/Window.hpp>
10
11 #include <math.h>
12 #include <limits.h>
13
14 #include <iostream>
15 #include <string>
16 #include <exception>
17 #include <stdexcept>
18 #include <vector>
19
20 #include "CircleBuffer.h"
21 #include "StringSound.h"
22
23 #define CONCERT_A 220.0
24 #define SAMPLES_PER_SEC 44100
25
26 using namespace std;
27 using namespace sf;
28

```

```

29 std::vector<sf::Int16> makeSamples(StringSound ss) {
30     std::vector<sf::Int16> samples;
31
32     ss.pluck();
33     int duration = 8;    // seconds
34     int i;
35     for (i = 0; i < SAMPLES_PER_SEC * duration; i++) {
36         ss.tic();
37         samples.push_back(ss.sample());
38     }
39     return samples;
40 }
41
42
43 int main() {
44     sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Plucked
        String Sound Lite");
45     sf::Event event;
46     double freq;
47     std::vector<sf::Int16> sample;
48
49     //lambda
50     auto return_freq = [](int n) { return 440 * pow(2, ((n - 24.0) /
        12.0)); };
51
52     std::vector<std::vector<int16_t>>samples; //vector of 37 vector
        samples
53     std::vector<sf::SoundBuffer> soundBuffers;
54     std::vector<sf::Sound> sounds;
55     for (int i = 0; i < 37; i++) {
56         freq = return_freq(i);
57         StringSound save_freq = StringSound(freq);
58         sample = makeSamples(save_freq);
59         samples.push_back(sample);    //pushback 37 samples
60     }
61     //std::vector<sf::SoundBuffer> soundBuffers; //37 samples go
        into soundBuffers
62     for (int i = 0; i < 37; i++) {
63         sf::SoundBuffer buffers;
64         buffers.loadFromSamples(&samples.at(i)[0], samples.at(i).
            size(), 2, SAMPLES_PER_SEC);
65         soundBuffers.push_back(buffers);
66     }
67     //std::vector<sf::Sound> sounds; //37 sounds go into soundBuffers
68     for (int i = 0; i < 37; i++) {
69         sf::Sound sound;
70         sound.setBuffer(soundBuffers[i]);
71         sounds.push_back(sound);

```

```

72     }
73
74     std::string keyboard = "q2we4r5ty7u8i9op-[]=zxdcfvgnbjmk,.;/' ";
    //make a keyboard string
75     while (window.isOpen()) {
76         while (window.pollEvent(event)) {
77             switch (event.type) {
78                 case sf::Event::Closed:
79                     window.close();
80                     break;
81                 case sf::Event::TextEntered: {
82                     string save_string;
83                     save_string = save_string + event.text.unicode; //
                        encoding characters in string
84                     int n = keyboard.find(save_string); //find the key
                        within the string
85                     if (n < 0) {
86                         break; //exit if it is the wrong key
87                     }
88                     sounds[n].play();
89                 }
90                 default:
91                     break;
92             }
93             window.clear();
94             window.display();
95         }
96     }
97     return 0;
98 }

```

## 7.6 CircleBuffer.h

I left this file the same as I did in part A of this assignment. In my header file, I decided to write CircularBuffer in a different way, I only had three integer variables, the capacity, front, and the bufferSize. I figured that I could figure out the back of the queue with the front variable. I had also used a unique pointer of `int16_t` because it would not resize efficiently as a regular vector.

```

1 // Copyright 2020 Stephanie La
2
3 #ifndef CIRCLEBUFFER_H // NOLINT
4 #define CIRCLEBUFFER_H // NOLINT
5 // #ifndef _HOME_SLAW_PS3A_CIRCULARBUFFER_H_
6 // #define _HOME_SLAW_PS3A_CIRCULARBUFFER_H_
7 #include <stdint.h>
8 #include <iostream>
9 #include <vector>
10 #include <memory>
11

```

```

12 class CircularBuffer {
13 public:
14     // create an empty ring buffer, with given max
15     explicit CircularBuffer(int capacity);
16     // capacity
17     // return number of items currently in the buffer
18     int size();
19     // is the buffer empty (size equals zero)?
20     bool isEmpty();
21     // is the buffer full (size equals capacity)?
22     bool isFull();
23     // add item x to the end
24     void enqueue(int16_t x);
25     // delete and return item from the front
26     int16_t dequeue();
27     // return (but do not delete) item from the front
28     int16_t peek();
29     void emptyBuffer();
30
31 private:
32     std::unique_ptr<int16_t[]>buffer;
33     int bufferSize;
34     int bufferCapacity;
35     int front;
36 };
37 #endif // _HOME_SLAW_PS3A_CIRCULARBUFFER_H_

```

## 7.7 StringSound.h

In the header file, I just included the following API and included two private members: the tick count and a CircularBuffer Pointer to store data into as per instructions.

```

1 #ifndef STRINGSOUND_H
2 #define STRINGSOUND_H
3
4 #include "CircleBuffer.h"
5
6 #include <stdint.h>
7 #include <iostream>
8 #include <vector>
9 #include <memory>
10
11 const int SAMPLING_RATE = 44100;
12 const double DECAY_FACTOR = 0.996;
13
14 class StringSound {
15 public:

```

```

16     StringSound(double frequency);           // create a guitar
        string sound of the
17                                           // given frequency
                                           // using a sampling
                                           // rate
18                                           // of 44,100
19     StringSound(std::vector<int16_t> init);   // create a guitar
        string with
20                                           // size and initial
                                           // values are given
                                           // by
21                                           // the vector
22     void pluck();                             // pluck the guitar
        string by replacing
23                                           // the buffer with
                                           // random values,
24                                           // representing
                                           // white noise
25     void tic();                               // advance the
        simulation one time step
26     int16_t sample();                         // return the
        current sample
27     int time();                              // return number of
        times tic was called
28                                           // so far
29 private:
30     CircularBuffer* buffer;
31     int tick_count;
32 };
33 #endif //STRINGSOUND_H

```

## 7.8 CircleBuffer.cpp

I left this file the same as I did in part A of this assignment. In this cpp file, I implemented calculations and made a temporary variable called back to keep track of the calculation to mod the buffercapacity and then saved the result and incremented the size since it was a fixed array. Many of the functions wanted to check if it was full or not and throw an error if it was not full or empty.

```

1 // Copyright 2020 Stephanie La
2
3 #include "CircleBuffer.h"
4 // #include "/home/slaw/ps3a/CircularBuffer.h"
5 #include <stdint.h>
6 #include <iostream>
7 #include <stdexcept>
8 #include <vector>
9 #include <memory>
10

```



```

11 CircularBuffer::CircularBuffer(int capacity) {
12     if (capacity < 1) {
13         throw std::invalid_argument(
14             "CB constructor: capacity must be greater than zero");
15     }
16     bufferCapacity = capacity;
17     bufferSize = 0;
18     front = 0;
19     buffer = std::unique_ptr<int16_t[]>(new int16_t[capacity]);
20 }
21
22 int CircularBuffer::size() {
23     return bufferSize;
24 }
25
26 bool CircularBuffer::isEmpty() {
27     return bufferSize == 0;
28 }
29
30 bool CircularBuffer::isFull() {
31     return bufferSize == bufferCapacity;
32 }
33
34 void CircularBuffer::enqueue(int16_t x) {
35     if (isFull()) {
36         throw std::runtime_error("enqueue: can't enqueue to a full
37             ring.");
38     }
39     // back declaration
40     int back = (front + bufferSize) % bufferCapacity;
41     buffer[back] = x;
42     bufferSize++;
43 }
44
45 int16_t CircularBuffer::dequeue() {
46     if (isEmpty()) {
47         throw std::runtime_error("Buffer is empty");
48     }
49     // save
50     int16_t temp = buffer[front];
51     front = (front + 1) % bufferCapacity;
52     bufferSize--;
53     return temp;
54 }
55
56 int16_t CircularBuffer::peek() {
57     if (isEmpty()) {
58         throw std::runtime_error("Buffer is empty");
59     }

```

```

58     }
59     return buffer[front];
60     // front is at index front
61     // holds the index where the data begins
62     // return buffer.at(front); //peek "front" of vector buff also
        valid
63 }
64
65 void CircularBuffer::emptyBuffer() {
66     front = 0;
67     bufferSize = 0;
68 }

```

## 7.9 StringSound.cpp

In this header file, I had to check for a couple of things: whether the frequency was empty and if my vector was empty and if not, an error message would be thrown. In this file, I had write my constructor where it would actively push zeros into the vector to create it. I would also have to find the result of the pluck() by writing calculations to find the frequency. I had to do a bit of research to find this formula, to implement the correct guitar sound. I also saved the decay rate to a constant int so I could use it without typing the number over and over again.

```

1  #include  "StringSound.h"
2  #include  "CircleBuffer.h"
3
4  #include <stdint.h>
5  #include <cmath>
6  #include <iostream>
7  #include <stdexcept>
8  #inc#include  "StringSound.h"
9  #include  "CircleBuffer.h"
10
11 #include <stdint.h>
12 #include <cmath>
13 #include <iostream>
14 #include <stdexcept>
15 #include <vector>
16 #include <random>
17
18
19 StringSound::StringSound(double frequency) {
20     //check if frequency is less than zero
21     if (frequency < 0) {
22         throw std::invalid_argument(
23             "frequency must be greater than zero");
24     }
25     int N = ceil(SAMPLING_RATE / frequency);
26     buffer = new CircularBuffer(N); //pass in new object

```

```

27     for (int i = 0; i < N; i++) {
28         buffer->enqueue(0);
29     }
30     tick_count = 0;
31 }
32
33 StringSound::StringSound(std::vector<int16_t> init) {
34     //check if vector is empty
35     if (init.empty()) {
36         throw std::invalid_argument(
37             "vector must be empty");
38     }
39     buffer = new CircularBuffer(init.size());
40     for (int i = 0; i < init.size(); i++) {
41         buffer->enqueue(init.at(i));
42     }
43     tick_count = 0;
44 }
45
46 void StringSound::pluck() {
47     // Set up randomness
48     std::random_device rd;
49     while (!buffer->isEmpty()) { //while not empty, clear buffer
50         buffer->dequeue();
51     }
52     while (!buffer->isFull()) { //while the buffer is not full, fill
53         //the buffer with random numbers
54         int16_t result = (rd() % (32767 - (-32768) + 1)) + (-32768);
55         //result = rd()%((max- min) + 1) + min;
56         buffer->enqueue(result);
57     }
58 }
59
60 void StringSound::tic() {
61     //lambda
62     auto return_enqueue = [&](int16_t x) { return buffer->enqueue(x)
63         ; };
64
65     int16_t step = ((buffer->dequeue() + buffer->peek()) / 2) *
66         DECAY_FACTOR;
67     return_enqueue(step);
68     //buffer->enqueue(step);
69     tick_count++;
70 }
71
72 int16_t StringSound::sample() {
73     return buffer->peek();
74 }

```

```

71 }
72
73 int StringSound::time() {
74     return tick_count;
75 }
76 #include <vector>
77
78
79 StringSound::StringSound(double frequency) {
80     //check if frequency is less than zero
81     if (frequency < 0) {
82         throw std::invalid_argument(
83             "frequency must be greater than zero");
84     }
85     int N = ceil(SAMPLING_RATE / frequency);
86     buffer = new CircularBuffer(N);
87     for (int i = 0; i < N; i++) {
88         buffer->enqueue(0);
89     }
90     tick_count = 0;
91 }
92
93 StringSound::StringSound(std::vector<int16_t> init) {
94     //check if vector is empty
95     if (init.empty()) {
96         throw std::invalid_argument(
97             "vector must be empty");
98     }
99     buffer = new CircularBuffer(init.size());
100     for (int i = 0; i < init.size(); i++) {
101         buffer->enqueue(init.at(i));
102     }
103     tick_count = 0;
104 }
105
106 void StringSound::pluck() {
107     // Set up randomness
108     std::random_device rd;
109     while (!buffer->isEmpty()) { //while not empty, clear buffer
110         buffer->dequeue();
111     }
112     while (!buffer->isFull()) { //while the buffer is not full, fill
        the buffer with random numbers
113         int16_t result = (rd() % (32767 - (-32768) + 1)) + (-32768);
114         buffer->enqueue(result);
115     }
116 }
117 }

```

```

118
119 void StringSound::tic() {
120     int16_t step = ((buffer->dequeue() + buffer->peek()) / 2) *
        DECAY_FACTOR;
121     buffer->enqueue(step);
122     tick_count++;
123 }
124
125 int16_t StringSound::sample() {
126     return buffer->peek();
127 }
128
129 int StringSound::time() {
130     return tick_count;
131 }

```

### 7.10 Sound result

The screen is black, but when running this code, music will be heard.

## 8 Ps4 - DNA Sequence Alignment

### 8.1 Overview

The goals of this assignment are to solve a problem in computational biology and to learn about a programming paradigm known as dynamic programming. In this assignment we had to calculate the edit distance between two strings. This has crucial in genetics and biology because it can be used to compare strings of DNA to find similarities or differences.

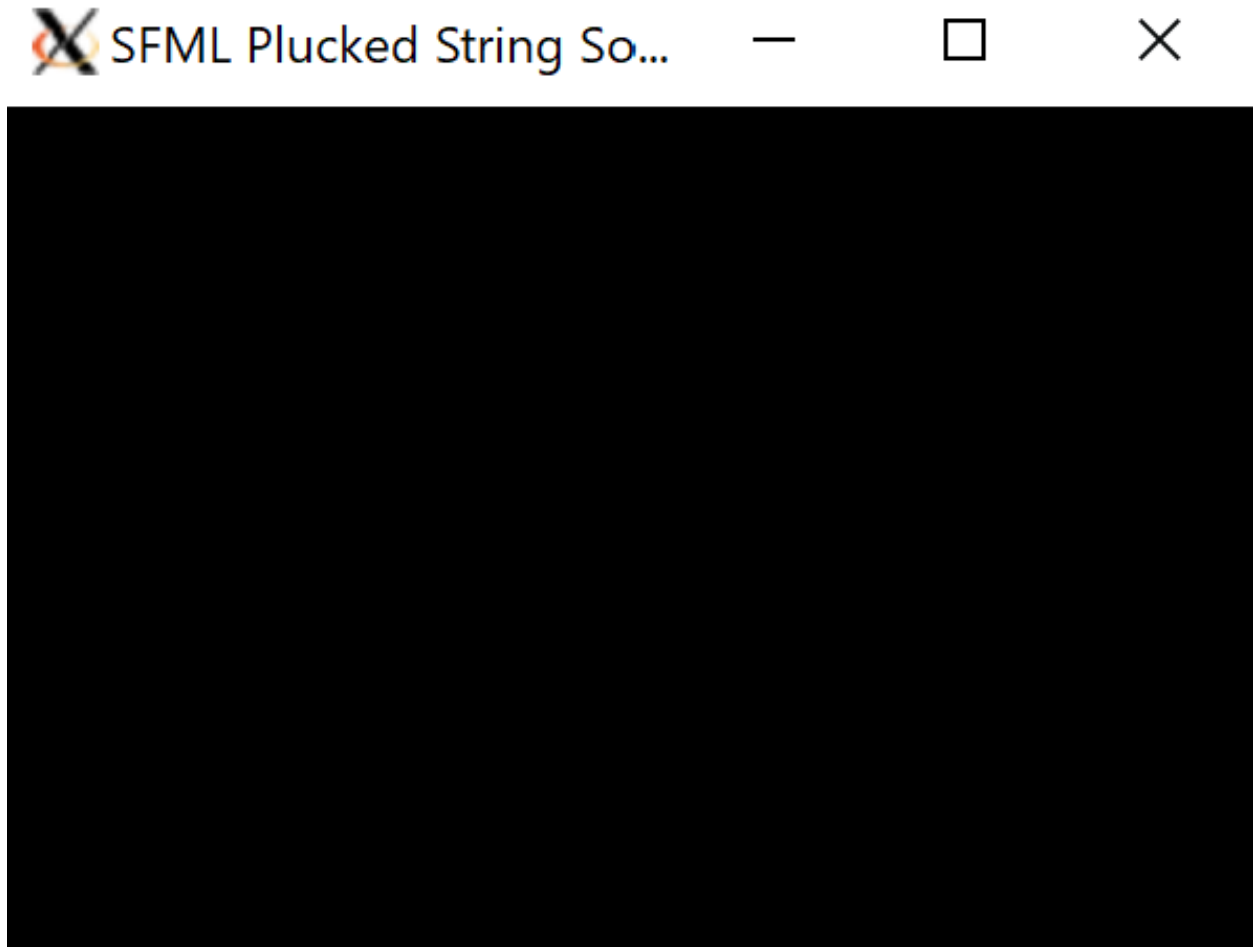
### 8.2 Key Concepts

The key concept in this assignment was to learn about dynamic programming and Valgrind, a really powerful debugging tool. I had saved the lengths of each string in two strings and often compared them to create a penalty chart if there was a letter missing, matched, or mismatched. Another key concept is using a lambda expression to make code more efficient and faster to run at compile time.

### 8.3 Learnings and Accomplishments

I had to learn about dynamic programming which is a technique I had never used before. It seemed like a small step by step problem solving technique meant for bigger projects, to store answers and use them for bigger implementation. Using a lambda expression was one of the challenges I faced throughout this assignment. I slowly learned that you can use a lambda expression to do calculations, where a function could substitute condensed code. I used dynamic programming as my implementation to write my program as a result. I chose this approach because since we have to fill a table, I needed to break it down into smaller problems, where I would fill out the outer column first with gap penalties, and then fill the insides. I get to store the results of the subproblems from another function and use them for another part of the program.

Figure 10: Screenshot of a window that would play music



The pros of using dynamic programming is that you can use it to work on multi-step programs and save results so other sub problems can reuse them to solve the original problem. This avoids recomputing the same quantity over and over again. The Needleman and Wunsch method provides good time and memory complexity, and a multi linear time complexity  $O(mn)$ . The cons of this method is really that it just a lot of memory is used because we divide problems into sub problems.

## 8.4 Makefile

The makefile was the same format but only the SFML system was used in this case. I had to also add `-std=c++11` in order to use a lambda expression.

```
1 CC = g++
2 CFLAGS = -g -Wall -Werror -ansi -pedantic -std=c++11 -O3
3
4 ED: EditDistance.o main.o
5     $(CC) $(CFLAGS) -o ED EditDistance.o main.o -lsfml-system
6 all: ED
7 EditDistance.o: EditDistance.cpp EditDistance.h
```

Figure 11: Numerical Score Table

operation	cost
insert a gap	2
align two characters that mismatch	1
align two characters that match	0

```

8      $(CC) $(CFLAGS) -c EditDistance.cpp
9 main.o: main.cpp
10     $(CC) $(CFLAGS) -c main.cpp
11 clean:
12     rm *.o ED

```

### 8.5 main.cpp

In the main.cpp file, I made a clock and time object and passed two strings in with each respective length. I then made an EditDistance object and passed the strings in. I then created an variable called opt where I would store the result of the EditDistance Class OptDistance() into. I did the same to the Alignment() function. I then printed out the edit distance total at the end and the execution time for the program.

```

1 // Copyright 2020 Stephanie La
2
3 #include <SFML/System.hpp>
4
5 #include "EditDistance.h"
6 #include <iostream>
7 #include <string>
8
9
10 int main() {
11     sf::Clock clock;
12     sf::Time t;
13
14     // passing in two strings
15     std::string one;
16     std::string two;
17
18     std::cin >> one;
19     std::cin >> two;
20
21     // make an object
22     EditDistance ED(one, two);
23
24     // initialize two functions
25     int opt = ED.OptDistance();
26

```

```

27     std::string align = ED.Alignment();
28
29
30     std::cout << "Edit distance = " << opt << std::endl;
31     std::cout << align << std::endl;
32
33     t = clock.getElapsedTime();
34     std::cout << "Execution time is " << t.asSeconds() << " seconds
        \n";
35
36
37     return 0;
38 }
39 // At the end of main, after computing the solution, capture the
    running time:
40 // t = clock.getElapsedTime();
41 // Then after printing out the solution, display the running time:
42 // cout << "Execution time is " << t.asSeconds() << " seconds \n";

```

## 8.6 EditDistance.h

In this header file, I decided to practice making constant integers just like macros. I made the GAP penalty equal to 2 since matching letters was a penalty of 0 and mismatched was 1. I made a vector of vector of integers to create a dynamically allocated matrix that could resize automatically with the removal or addition of data. My only other additional private members were just two strings.

```

1 // Copyright 2020 Stephanie La
2
3
4 #ifndef EDITDISTANCE_H // NOLINT
5 #define EDITDISTANCE_H // NOLINT
6
7 #include <SFML/System.hpp>
8 #include <iostream>
9 #include <vector>
10 #include <string>
11 #include <sstream>
12 #include <algorithm> // NOLINT
13
14 const int GAP = 2;
15
16 class EditDistance {
17 public:
18     EditDistance(std::string a, std::string b);
19     static int penalty(char a, char b);
20     static int min(int a, int b, int c);
21     int OptDistance();

```



```

22  std::string Alignment();
23  ~EditDistance();
24
25  private:
26      // input strings
27      std::string one;
28      std::string two;
29
30      // dynamic array to resize automatically
31      // std::vector< std::vector< int > > data;
32      std::vector< std::vector< int > > matrix;
33 };
34 #endif // NOLINT

```

## 8.7 EditDistance.cpp

In this cpp file, I left a comments in order to follow the long steps of dynamic programming. I had also implemented a lambda called return push, where it would push back a zero into the matrix to create the spaces. I then looped through the constructor using that lambda expression and then storing it in a temp value. I had used if and else statements to determine the minimum of the three penalties and whether it would be a zero or one depending on the alignment of the characters. In OptDistance(), I used the algorithm to fill the bottom row and last column up with the penalty points first, and then filled in the inner matrix with the min() and the penalty() functions. With the alignment() function, I had to make sure the matrix didn't fall out of bounds and cut off the strings, so I had used a ternary operator to return a -1 if it did occur. Alignment was to align the letters at best fit depending on the penalties that accumulated. The problem was to trying to align the different dashes that involved more scrutiny than just spacing.

```

1 // Copyright 2020 Stephanie La
2
3 // #include "/home/slaw/ps4/EditDistance.h" // NOLINT
4
5 // this should be the write header file below
6 #include "EditDistance.h"
7
8 #include <string>
9 #include <vector>
10 #include <algorithm> // NOLINT
11
12
13 // constructor
14 EditDistance::EditDistance(std::string a, std::string b) {
15     // accepts the two strings to be compared
16     one = a;
17     two = b;
18
19     // temp vector to use to store
20     std::vector<int> temp;

```

```

21
22 // lambda
23 auto return_push = [&](int n) { temp.push_back(0); };
24
25 // fill matrix with 0's
26 for (unsigned j = 0; j < two.length() + 1; j++) {
27     return_push(0);
28     // temp.push_back(0);
29 }
30 // fill the other string with the temp
31 for (unsigned i = 0; i < one.length() + 1; i++) {
32     matrix.push_back(temp);
33 }
34 }
35
36 // destructor
37 EditDistance::~EditDistance()
38 {}
39
40 int EditDistance::penalty(char a, char b) {
41     /* returns the penalty for aligning chars a
42        and b(this will be a 0 or a 1).*/
43     if (a == b) {
44         return 0;
45     } else {
46         return 1;
47     }
48 }
49
50 int EditDistance::min(int a, int b, int c) {
51     /* returns the minimum of the three
52        arguments.*/
53     if (a <= b && a <= c) {
54         return a;
55     } else if (b <= c) {
56         return b;
57     } else {
58         return c;
59     }
60 }
61
62 int EditDistance::OptDistance() {
63     /* populates the matrix based on having the two strings, and
64        returns the optimal distance(from the[0][0] cell of the
65        matrix when done).*/
66
67     // two indexes for two strings
68     int i = 0;

```

```

68  int j = 0;
69  // initializing strings
70  int string_one_length = one.length();
71  int string_two_length = two.length();
72
73  // fill bottom row and last column with gap penalty
74  for (i = string_one_length; i >= 0; i--) {
75  matrix[i][string_two_length] = GAP * (string_one_length - i);
76  }
77  for (j = string_two_length; j >= 0; j--) {
78  matrix[string_one_length][j] = GAP * (string_two_length - j);
79  }
80
81  // fill inside matrix with ints in every row and column
82  // passing the matrix the min value of the 3 values
83  for (i = (string_one_length - 1); i >= 0; i--) {
84  for (j = (string_two_length - 1); j >= 0; j--) {
85  int first = GAP + matrix[i + 1][j];
86  int second = GAP + matrix[i][j + 1];
87  int third = matrix[i + 1][j + 1] + penalty(one[i], two[j]);
88  matrix[i][j] = min(first, second, third);
89  }
90  }
91  return matrix[0][0];
92 }
93
94 std::string EditDistance::Alignment() {
95  // traces the matrix and returns a string that can be printed
96  // to display the actual alignment. In general, this will be a
    multi
97  // line string with embedded \n's
98
99  // get alignment from top left to bottom right
100
101
102  // inherited from ostream functionality, object oriented
103  std::stringstream return_string;
104
105  // my indexes for two strings
106  int i = 0;
107  int j = 0;
108
109  // accessing strings
110  // string one represents columns
111  // string two represents rows
112  int string_one = one.length();
113  int string_two = two.length();
114

```

```

115 // while matrix is not filled
116 while (i < string_one || j < string_two) {
117     int current_position = matrix[i][j];
118
119     // if i < string_one {return GAP + matrix[i + 1][j] } else {return
        -1}
120     // same as second var
121     int first = i < string_one ? GAP + matrix[i + 1][j] : -1;
122     int second = j < string_two ? GAP + matrix[i][j + 1] : -1;
123     int penalty_result = penalty(one[i], two[j]);
124
125     // down
126     if (current_position == first) {
127         return_string << one[i] << " - " << "2\n";
128         i++;
129     } else if (current_position == second) {
130         // right
131         return_string << "- " << two[j] << " 2\n";
132         j++;
133     } else {
134         // diagonal
135         return_string << one[i] << " " << two[j] << " " << penalty_result
            << " \n";
136         i++;
137         j++;
138     }
139 }
140 string result_string = return_string.str();
141 return result_string;
142 }

```

## 8.8 Execution Result

# 9 Ps5 - Markov Model of Natural Language

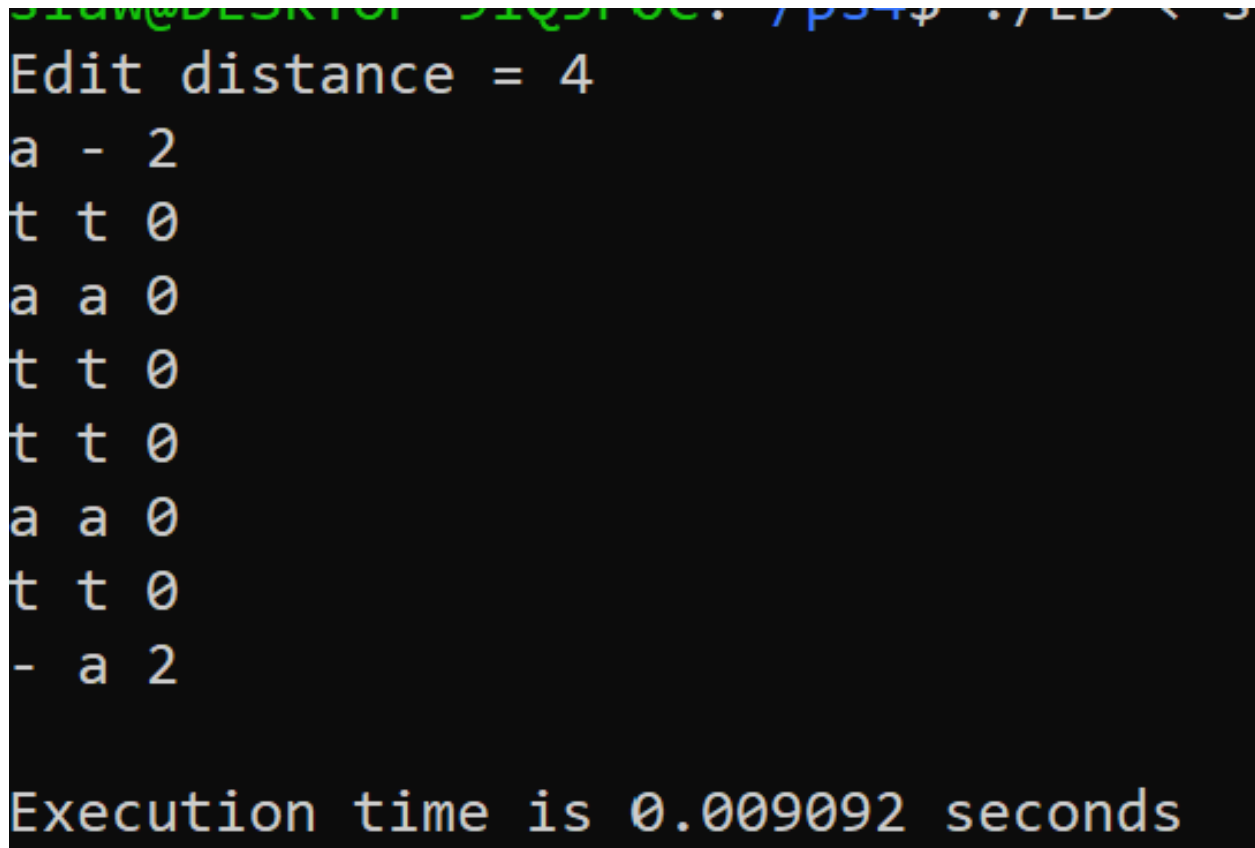
## 9.1 Overview

In this assignment we had to create a markov model to process text and generate novel but natural looking output text based on the source.

## 9.2 Key Concepts

A key data structure used for this assignment was `std::map` which was used to create associations between string kgrams and information about their frequency in the text to implement the Markov model.

Figure 12: Execution time with DNA strands

A terminal window with a black background and yellow text. The text displays the results of a DNA strand analysis. At the top, it says 'Edit distance = 4'. Below this, there are several lines of DNA sequences: 'a - 2', 't t 0', 'a a 0', 't t 0', 't t 0', 'a a 0', 't t 0', and '- a 2'. At the bottom, it states 'Execution time is 0.009092 seconds'.

```
Edit distance = 4
a - 2
t t 0
a a 0
t t 0
t t 0
a a 0
t t 0
- a 2

Execution time is 0.009092 seconds
```

### 9.3 Learnings and Accomplishments

I did not have much experience with `std::map` so I had to learn how to use it to complete this assignment. I also was not able to finish the assignment parts with `TextGenerator.cpp` and the `test.cpp` file. However, I know that the end result will be a file full of words and the text generator will generate each kgram and how often each character follows each group of kgrams.

### 9.4 Makefile

The makefile was the same format but only I also had to add the boost unit test framework. I did not use any of the SFML libraries this time.

```
1 CC = g++
2 CFLAGS = -g -w -Wall -Werror -ansi -pedantic -std=c++11
3
4 TextGenerator: MModel.o test.o
5     $(CC) $(CFLAGS) -o TextGenerator MModel.o test.o -
6         lboost_unit_test_framework
7 all: TextGenerator
8 MModel.o: MModel.cpp MModel.h
9     $(CC) $(CFLAGS) -c MModel.cpp
10 test.o: test.cpp
11     $(CC) $(CFLAGS) -c test.cpp
```

```

11 clean:
12     rm *.o TextGenerator

```

## 9.5 MModel.h

In my header file, I tried to use the map with a vector of strings and key pair of integers and store the results in a string called alphabet and freq char to store the letters that would frequently appear after the kgram itself.

```

1 // Copyright 2020 Stephanie La
2 #ifndef MMODEL_H // NOLINT
3 #define MMODEL_H // NOLINT
4
5
6 #include <iostream>
7 #include <map>
8 #include <vector>
9 #include <string> // NOLINT
10 #include <exception>
11
12
13 using std::string;
14 using std::map;
15 using std::ostream;
16 using std::vector;
17 using std::endl;
18
19
20 // Note: all of the below constructors/methods should be public.
21 class MModel {
22 public:
23     // create a Markov model of order k from given text
24     MModel(string text, int k);
25     // Assume that text has length at least k
26     int kOrder();
27     // number of occurrences of kgram in text
28     int freq(string kgram);
29     // (throw an exception if kgram is not of length k
30     int freq(string kgram, char c);
31     char kRand(string kgram);
32     string generate(string kgram, int L);
33     friend ostream& operator<<(ostream& os, MModel& mm);
34 private:
35     int order;
36     // need one int for every kgram to keep track of how
37     // many times you saw it
38     map<string, int> kgrams;
39     string alphabet;

```

```

40 // collects frequency of the letter after the k grams
41 string freq_char;
42 // of times it was followed by a specific character
43 };
44 #endif // MMODEL_H // NOLINT

```

## 9.6 MModel.cpp

In this cpp file, I did not manage to finish the constructor or overload the ostream function. Most of the functions required me to throw an runtime error to make sure the kgram either existed or wasn't the same length as the integer L. If not, I would store the letters into the map and then return it to the function.

```

1 // Copyright 2020 Stephanie La
2 #include "MModel.h"
3 #include <string> // NOLINT
4 #include <map> // NOLINT
5
6 // Assume that text has length at least k
7 MModel::MModel(std::string text, int k) {
8     int i;
9     string new_gram;
10    // text length has length at least to k
11    for (int i = 0; i < text.size() - k; i++) {
12        new_gram = text.substr(i, k);
13        kgrams[new_gram]++;
14        std::size_t my_char = alphabet.find(text.at(i));
15    }
16 }
17
18 // order k of Markov model
19 int MModel::kOrder() {
20     return order;
21 }
22
23 // number of occurrences of kgram in text
24 int MModel::freq(string kgram) {
25     if (kgram.size() != order) {
26         throw std::runtime_error("kgram is not of length k");
27     }
28     return kgrams[kgram];
29 }
30
31
32 // number of times that character c follows kgram
33 // if order=0, return num of times char c appears
34 // (throw an exception if kgram is not of length k)
35 int MModel::freq(string kgram, char c) {

```

```

36  if (kgram.size() != order) {
37  throw std::runtime_error("kgram is not of length k");
38  }
39  if (order == 0) {
40  // index to count the num of times c appears
41  int count = 0;
42  for (int i = 0; i < freq_char.size(); i++) {
43  if (freq_char[i] == c) {
44  // increment count index
45  count++;
46  return count;
47  } else {
48  // return char with kgram map
49  return kgrams[kgram + c];
50  }
51  }
52  }
53 }
54
55 // random character following given kgram
56 // (Throw an exception if kgram is not of length k.
57 // Throw an exception if no such kgram.)
58 char MModel::kRand(string kgram) {
59  if (kgram.size() != order) {
60  throw std::runtime_error("kgram is not of length k");
61  // check if kgrams exist or not
62  } else if (kgram.size() == 0) {
63  throw std::runtime_error("Kgram does not exist");
64  }
65  // temp string var to push rand number of 0 and kgrams
66  string temp;
67  for (int i = 0; i < alphabet.size(); i++) {
68  for (int n = 0; n < kgrams[kgram + alphabet[i]]; n++) {
69  temp.push_back(alphabet[i]);
70  }
71  }
72  return temp[rand() % temp.size()];
73 }
74
75 // generate a string of length L characters
76 // by simulating a trajectory through the corresponding
77 // Markov chain. The first k characters of the newly
78 // generated string should be the argument kgram.
79 // Throw an exception if kgram is not of length k.
80 // Assume that L is at least k.
81 string MModel::generate(string kgram, int L) {
82  int size = kgram.size();
83  if (size != kOrder()) {

```



```

84  throw std::runtime_error("kgram is not of length k");
85  } else {
86  // a string of kgram to iterate through L
87  string result_kgram = kgram;
88  int i = 0;
89  while (result_kgram.size() < L) {
90  // temp vars to store the result
91  string temp;
92  char l;
93  // create a substring for kgrams generated
94  // store temp into a char and push into result string
95  temp = result_kgram.substr(i, order);
96  l = kRand(temp);
97  result_kgram.push_back(l);
98  i++;
99  }
100 return result_kgram;
101 }
102 }
103
104 // << // overload the stream insertion operator and display
105 // the internal state of the Markov Model. Print out
106 // the order, the alphabet, and the frequencies of
107 // the k-grams and k+1-grams.
108 ostream& operator<<(ostream& os, MModel& mm) {
109  os << "The order is " << mm.order << endl;
110  os << "The alphabet is " << mm.alphabet << endl;
111  map<string, int>::iterator ptr;
112 }

```

## 10 Ps6 - Kronos Time Clock: Introduction to Regular Expression Parsing

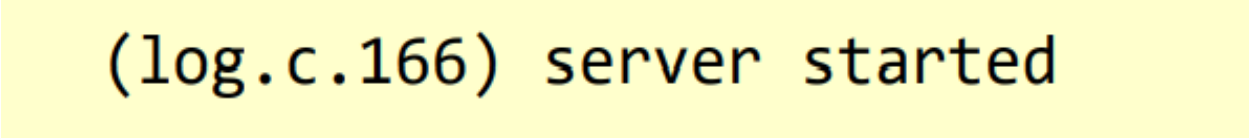
### 10.1 Overview

This next project was about parsing a the Kronos InTouch time clock by using regular expressions. This project takes a log file in and outputs an .rpt file. I used one big regex string, separating it into groups. If the regex matches the string show, it will display a success message if the bootup was complete, and how long it took to start boot. Otherwise, it would output a message saying "Incomplete boot." When an InTouch device boots up, the first logging message that will be seen is: When the bootup is completed, this message will be displayed:

### 10.2 Key Concepts

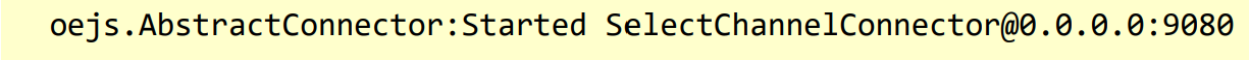
The key concept in this assignment was to help us learn what regex expressions were and how to use them to find strings within a bunch of words in a file. It also taught us how to use regex libraries and date and time libraries to keep track of time in our program.

Figure 13: Example Start Message



(log.c.166) server started

Figure 14: Example End Message



oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080

### 10.3 Learnings and Accomplishments

I learned that utilizing regex expressions are part of parsing strings and are often used to find words and letters in a heavily dense document or file. I have used regex expressions when I was on my internship during the spring of 2020. We would have to parse files and look for strings from URLs that would appear in the project scripts. My approach for solving this problem started when I had to figure out how regular expressions worked and how to figure out the format to group a couple of numbers together. After learning how to read regex, I just needed to parse the log files, store and compare the lines the regex, and then write the output through the output file. I set up the file streams for tested the input and output files at first. I then had to figure out how regex match worked and it had to be the entire line it had to match. The program would output whether it was booted up successfully or not.

### 10.4 Makefile

The makefile was the a bit different without the libraries graphics, window, and system used from the SFML library. I also included the two libraries boost regex and boost date and time to utilize to utilize the time.

```
1 CC = g++
2 CFLAGS = -g -Wall -Werror -ansi -pedantic -std=c++11
3 LIBS = -lboost_regex -lboost_date_time
4
5 ps6: main.o
6     $(CC) $(CFLAGS) -o ps6 main.o $(LIBS)
7 all: ps6
8 main.o: main.cpp
9     $(CC) $(CFLAGS) -c main.cpp
10 clean:
11     rm *.o ps6
```

### 10.5 main.cpp

The main design I used was the boost regex. This held the regular expression I used. The regex was used as a sort filter for parsing the log file and figuring out where you are in the file in order to start the clock. This program outputs an rpt file that is standardized output that has the boot time in milliseconds and the date the boot happened on. I made sure that the program was given two arguments, the executable and the input file name, and throw an error if it fails. then, I had

checked to see if the file had opened or not, and would exit if it failed to open. I then proceeded to save the input and output files and then write the matching regex expressions. For example, I learned that `[0-9]4` is looking for 4 numbers from 0-9, `[0-9]1,2` is looking for one or two numbers from 0-9, `[0-9]++` is looking for numbers between 0-9 and `+` checks for more than 1 number, and `(.*log.c.166.*)` is looking for the matching string. I then make a string and a smatch variable that will take in the regex expression in the parameter, and parse the regex with different `sm[]` parts and convert the date and time from string to integers. After the conversion, I wrote the "Device Boot" message to indicate the bootup was successful and a bool for the bootup will be true. The same thing holds true if it matches the end of the bootup regex, it will display the message that the bootup is complete and will display the time of when it completed and how fast it completed. If the bootup did not work, it will display the "Incomplete Boot" message and the boolean for the bootup will be false. Otherwise, it will display a "Unexpected Boot" message.

```
1 // Copyright 2020 Stephanie La
2 #include <boost/regex.hpp>
3 #include <exception>
4 #include <iostream>
5 #include <fstream>
6 #include <cstdlib>
7 #include <string>
8 #include "boost/date_time/posix_time/posix_time.hpp"
9 #include "boost/date_time/gregorian/gregorian.hpp"
10
11
12 using boost::gregorian::date;
13 using boost::gregorian::from_simple_string;
14 using boost::gregorian::date_period;
15 using boost::gregorian::date_duration;
16
17 using boost::posix_time::ptime;
18 using boost::posix_time::time_duration;
19
20 using std::cin;
21 using std::cout;
22 using std::endl;
23 using std::string;
24
25 int main(int argc, char* argv[]) {
26     // read in entire log file and
27
28     // first check for two arguments
29     if (argc != 2) {
30         throw std::runtime_error("Ener a log file");
31     }
32     // ifstream takes 1st argument, accept filename
33     std::ifstream inputFile(argv[1]);
34     // check if opening files is successful
35     if (!inputFile.is_open()) {
```

```

36         cout << "Unable to open file" << endl;
37         exit(1);
38     }
39     string fileName = argv[1];
40     string outFile(fileName + ".rpt");
41     std::ofstream outputFile;
42     // lambda
43     auto open_file = [&](string n) { outputFile.open(outFile.c_str()
44         ); };
45     open_file(outFile.c_str());
46
47     // outputFile.open(outFile.c_str());
48
49     // login message & completetion
50     boost::regex start_boot("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})"
51         "[([0-9]++) : ([0-9]++) : ([0-9]++)" " (*.log.c.166.*)");
52     boost::regex end_boot("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})"
53         "[([0-9]++) : ([0-9]++) : ([0-9]++)"
54         " (*.oejs.AbstractConnector:Started SelectChannelConnector.*)");
55
56     // string to compare regex
57     string line;
58     // match variables
59     boost::smatch sm;
60     // did the file open correctly bool
61     bool bootup = false;
62     int line_number = 1;
63     ptime t1, t2;
64
65     // while file is open
66     while (getline(inputFile, line)) {
67         // while the string matches the regex start boot
68         if (regex_match(line, sm, start_boot)) {
69             // if bootup fails
70             if (bootup) {
71                 outputFile << "**** Incomplete Boot **** \n" << endl
72                     ;
73             }
74             // make a date with gregorian
75             date d(from_simple_string(sm[0]));
76             ptime temptime(d, time_duration(stoi(sm[4]),
77                 stoi(sm[5]), stoi(sm[6])));
78             // save temptime into a ptime var
79             t1 = temptime;
80             outputFile << "== Device Boot == " << endl;
81             outputFile << line_number << "(" << argv[1] << "): ";
82             // date

```

```

81         outputFile << sm[1] << "-" << sm[2] << "-" << sm[3] << "
            ";
82         // times
83         outputFile << sm[4] << ":" << sm[5] << ":" << sm[6] << "
            ";
84         outputFile << "Boot Start" << endl;
85         // everything must be displayed, then it becomes true
86         bootup = true;
87     } else if (regex_match(line, sm, end_boot)) {
88         if (bootup) {
89             date d(from_simple_string(sm[0]));
90             ptime temptime(d, time_duration(stoi(sm[4]),
91                 stoi(sm[5]), stoi(sm[6])));
92             t2 = temptime;
93             time_duration td = t2 - t1;
94             outputFile << line_number << "(" << argv[1] << "): "
                ;
95             outputFile << sm[1] << "-" << sm[2] << "-" << sm[3]
                << " ";
96             outputFile << sm[4] << ":" << sm[5] << ":" << sm[6]
                << " ";
97             outputFile << "Boot Completed" << endl;
98
99             outputFile << "\t" << "Boot Time: ";
100            outputFile << td.total_milliseconds() << "ms \n" <<
                endl;
101            bootup = false;
102        } else {
103            outputFile << "**** Unexpected Boot****\n" << endl;
104        }
105    }
106    line_number++;
107 }
108 return 0;
109 }

```

## 10.6 Rpt Files

Overall, I learned about regex expressions and how they can be used for parsing a file. I also learned about the Kronos InTouch time clock log by using regular expressions to parse the file and verifying device boot up timing.

Figure 15: device intouch.log.rpt

```
== Device Boot ==
435369(device1_intouch.log): 2014-03-25 19:11:59 Boot Start
435759(device1_intouch.log): 2014-03-25 19:15:02 Boot Completed
    Boot Time: 183000ms

== Device Boot ==
436500(device1_intouch.log): 2014-03-25 19:29:59 Boot Start
436859(device1_intouch.log): 2014-03-25 19:32:44 Boot Completed
    Boot Time: 165000ms

== Device Boot ==
440719(device1_intouch.log): 2014-03-25 22:01:46 Boot Start
440791(device1_intouch.log): 2014-03-25 22:04:27 Boot Completed
```

Figure 16: device3 intouch.log.rpt

```
== Device Boot ==
31063(device3_intouch.log): 2014-01-26 09:55:07 Boot Start
31176(device3_intouch.log): 2014-01-26 09:58:04 Boot Completed
    Boot Time: 177000ms

== Device Boot ==
31274(device3_intouch.log): 2014-01-26 12:15:18 Boot Start
**** Incomplete Boot ****

== Device Boot ==
31293(device3_intouch.log): 2014-01-26 14:02:39 Boot Start
31401(device3_intouch.log): 2014-01-26 14:05:24 Boot Completed
    Boot Time: 165000ms
```