Meet Kothari

COMP IV: Final Project Portfolio

Spring 2021

Table of Contents:

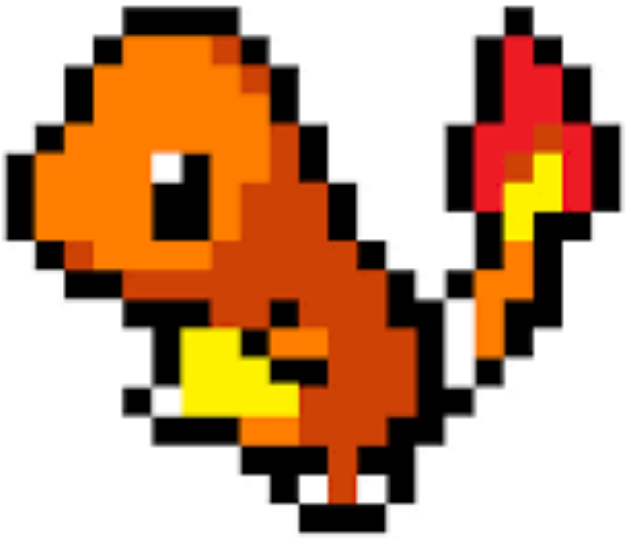
Time to complete: 8 hours.

PS0: Hello World With SFML

Purpose: The main purpose of this assignment was for me to get up and running with my build environment, as that would be needed in the coming assignments. Beyond that, I was instructed to successfully run the Hello World code provided by the SFML database. I was further instructed to extend the code. The assignment outlined that my program should be able to draw an image sprite, make the image sprite move, make the image sprite respond to keystrokes, and make it do something else of my choosing.

My version: My assignment correctly did all of these things, and for my extra function, I made it so that the sprite would rotate in place while it was being moved by the keystrokes, as per the images I've attached in the following page. As is evident, my original sprite of the beloved Pokémon, Charmander, was quite differently oriented by the end of the program.

What I learned: In this assignment, I was introduced to the basics of SFML which would carry me through the semester; outputting graphics to the screen, moving these graphics, etc. It did a wonderful job of laying down the foundational skill work. The greatest skill I learned from this precursory assignment was how to efficiently set up and open an SFML window and get things to display in it. I also learned a great deal about how to manipulate these things. As per the specifications of the program, however, I did not make a Makefile, as it was not required. Another thing that I learned was how in-depth the SFML library was, and how I'd need to allocate time to be able to finish the coming assignments well. There were no algorithms used.



Original Sprite

Output after being ran for a while

```
1: #include <SFML/System.hpp>
2: #include <SFML/Graphics.hpp>
3:
4: #include <iostream>
5:
6: using namespace std;
7:
8: int main(){
9:
10:         int x, y;
11:
12:         int movement = 8;
13:
14:         sf::Sprite sprite;
15:
16:         sf::Texture texture;
17:
18:         sf::Clock clock; // variable defining
19:
20:         sf::VideoMode desktopMode = sf::VideoMode::getDesktopMode(); // create a window
21:
22:         sf::RenderWindow window(sf::VideoMode(desktopMode.width desktopMode.height,
desktopMode.bitsPerPixel), "PS0");
23:
24:         if (!texture.loadFromFile("sprite.png")){ cout << "There was an error.” }
25:
26:     sprite.setTexture(texture); // apply the texture to the sprite
27:
28:     x = window.getSize().x/2; // coordinates for the sprite
29:
30:     y = window.getSize().y/2;
31:
32:     while (window.isOpen()){
33:
34:                 sf::Event event;
35:
36:                 while (window.pollEvent(event)){ // find out when a key is pressed
37:
```

```
38:                 if (event.type == sf::Event::Closed) window.close();
39:
40:                 if (event.type == sf::Event::KeyPressed){
41:
42:                         switch (event.key.code){ // move the sprite
43:
44:                            case sf::Keyboard::Escape : window.close(); break;
45:
46:                                 case sf::Keyboard::Up    : y-= movement; break;
47:
48:                                 case sf::Keyboard::Down  : y += movement; break;
49:
50:                                 case sf::Keyboard::Left  : x-= movement; break;
51:
52:                                 case sf::Keyboard::Right : x += movement; break;
53:
54:                                 default : break;
55:
56:                         }
57:
58:                 }
59:
60:           }
61:
62:           sprite.setPosition(x,y); // draw the sprite and rotate it
63:
64:           sprite.rotate(.01);
65:
66:           window.draw(sprite);
67:
68:           window.display(); // update the display
69:
70:         }
71:
72:         return 0;
73:
74:         }
75:
```

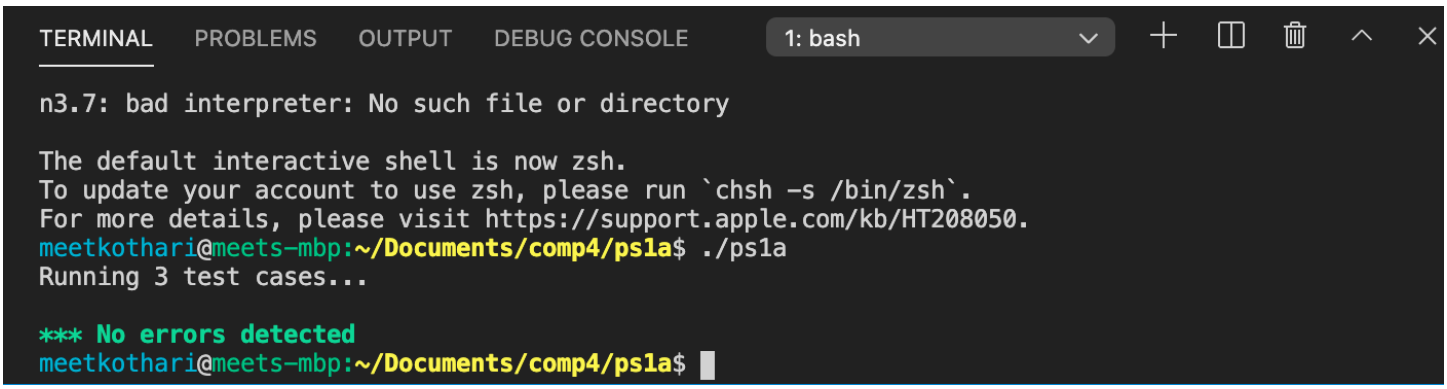
PS1: Linear Feedback Shift Register and Image Encoding

PS1A:

Purpose: This entire assignment centered around the creation of a program that would produce pseudo-random bits by simulating a linear feedback shift register, and then use it to implement a simple form of encryption for digital pictures.

My Version: For this portion of the assignment (PS1A), I implemented the FibLFSR class and unit tests using the Boost test framework. Basically, all of this was new knowledge to me, but the most difficult thing to wrap my head around at the start had been the idea of an LFSR. A linear feedback shift register (LFSR) is a register that takes a linear function of a previous state as an input- and that's what the majority of this part of the assignment centered around.

What I Learned: Along with the logic behind that, I learned a great deal about the testing process. I became intimate with Boost testing, learning quickly that it could allow a programmer to efficiently set up and run a series of test cases to prod at and run certain parts of the program. As per the screenshot in the next page, my test cases passed.



Makefile Sun Feb 07 20:48:24 2021 1

```
1: CC = g++
2: CFLAGS =-g -O -Wall -Werror -std=c++11 -pedantic
3: LIBS=-lboost_unit_test_framework
4:
5: all: ps1a
6: ps1a: test.o FibLFSR.o
7:         $(CC) $(CFLAGS) -o ps1a test.o FibLFSR.o $(LIBS)
8:
9: test.o: test.cpp FibLFSR.h
10:         $(CC) $(CFLAGS) -c test.cpp -o test.o
11:
12: FibLFSR.o: FibLFSR.cpp FibLFSR.h
13:         $(CC) $(CFLAGS) -c FibLFSR.cpp -o FibLFSR.o
14:
15: clean:
16:         rm ps1a test.o FibLFSR.o
```

FibLFSR.cpp Sun Feb 07 23:14:02 2021 1

```
1: #include <iostream>
2: #include "FibLFSR.h"
3:
4: using namespace std;
5:
6: FibLFSR::FibLFSR(string seed) { str = seed; } // initializes the seed
8: int FibLFSR::step(){ // function simulates one step of the LFSR
9:
10:     int compare;
11:     compare = 0;
12:
13:     byte3 = str.at(2) - '0';
14:     byte2 = str.at(3) - '0';
15:     byte1 = str.at(5) - '0';
16:     byte4 = str.at(0) - '0';
17:
18:     if ((byte4 == 1 && byte3 == 1) || (byte4 == 0 && byte3 == 0)){ compare = 0; }
19:     else{ compare = 1; }
20:     if (compare == byte2){ compare = 0; }
21:     else { compare = 1; }
```

```
22:     if (compare == byte1){ compare = 0; }
23:     else { compare = 1; }
24:
25:     str.erase(str.begin());        // make the bit spot at the beginning
open
26:     str.push_back(compare + '0');  // push back the output
27:     return compare; // return the result, which is the comparison
28: }
29:
30: int FibLFSR::generate(int k){
31:
32:     int i, j; // initialization variables
33:     j= 0;
34:
35:     for (i = 0; i < k; i++){
36:
37:
38:
39:     }
40:
41:     return j;
42: }
43:
44: FibLFSR:: ~FibLFSR(){ memory.clear(); } // deallocate any leftover memory
45:
46: ostream &operator <<(std::ostream &out, const FibLFSR &fibLFSR) {
47:       return out << fibLFSR.str;
48:   } // overload the << stream insertion operator
```

FibLFSR.h Sun Feb 07 23:10:12 2021 1

```
1: #ifndef FIBLFSR_H
2: #define FIBLFSR_H
3:
4: #include <iostream>
5: #include <string>
6: #include <vector>
7:
8: using namespace std;
9:
```

```
10: class FibLFSR {
11:
12:     public:
13:
14:     FibLFSR(string seed); // constructor to create LFSR
15:
16:     int step(); // simulate one step and return the new bit as 0 or 1
17:
18:     int generate(int k); // simulate k steps and return k-bit integer
19:
20:      ~FibLFSR(); // a destructor to deallocate memory from the constructor
21:
22:     friend std::ostream &operator<<(ostream &out, const FibLFSR &fibLFSR)
; // overloads the << stream insertion operator to display its current register
value in printable form
23:
24:     private :
25:
26:     string str; // string for comparison
27:
28:     vector <int> memory; // for de-allocating memory at the end
29:
30:     int byte1, byte2, byte3, byte4; // variables to match the diagram
31: };
32:
33: #endif //FIBLFSR_H
```

---

test.cpp Sun Feb 07 22:54:04 2021 1

```
1: #include <iostream>
2: #include <string>
3:
4: #include "FibLFSR.h"
5:
6: #define BOOST_TEST_DYN_LINK
7: #define BOOST_TEST_MODULE Main
8:
9: #include <boost/test/unit_test.hpp>
10:
11: BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps){ // the test case provided
```

```
12:
13:   FibLFSR l("1011011000110110");
14:   BOOST_REQUIRE(l.step() == 0);
15:   BOOST_REQUIRE(l.step() == 0);
16:   BOOST_REQUIRE(l.step() == 0);
17:   BOOST_REQUIRE(l.step() == 1);
18:   BOOST_REQUIRE(l.step() == 1);
19:   BOOST_REQUIRE(l.step() == 0);
20:   BOOST_REQUIRE(l.step() == 0);
21:   BOOST_REQUIRE(l.step() == 1);
22:
23:   FibLFSR l2("1011011000110110");
24:   BOOST_REQUIRE(l2.generate(9) == 51);
25:
26: }
27:
28: BOOST_AUTO_TEST_CASE(stepexample){ // a test meant to test the step funct
ion
29:   FibLFSR a("11101111111111111100");
30:   BOOST_REQUIRE(a.step() == 1);
31:   BOOST_REQUIRE(a.step() == 1);
32:   BOOST_REQUIRE(a.step() == 0);
33: }
34:
35: BOOST_AUTO_TEST_CASE(generateexample){ // a test meant to test the genera
te function
36:   FibLFSR test3("1100011011000011");
37:   BOOST_REQUIRE(test3.generate(5) == 6);
38: }
39:
```

PS1B:

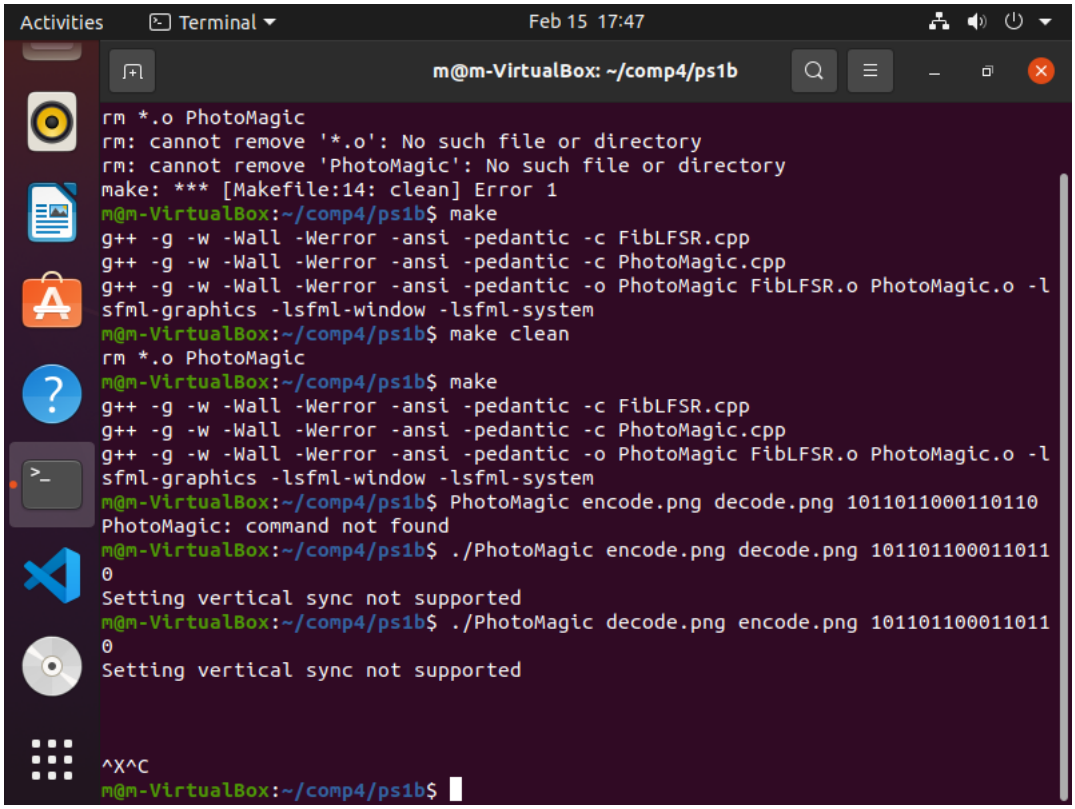
Purpose: To write a program *PhotoMagic.cpp* that could encrypt and decrypt pictures. This was done mainly through the use of the framework I'd laid down in the previous portion of the assignment, and a function called transform.

Key Algorithm: Transform was far and away the key algorithm in this particular assignment. The transform function worked by taking an image and an FibLFSR as arguments. It transformed the aforementioned image by using the linear feedback shift register. It basically shifted bits until the image was changed.

What I Learned: This was the trickiest part of the assignment, in particular, because it was completely different than any of the algorithms I'd developed in the past.

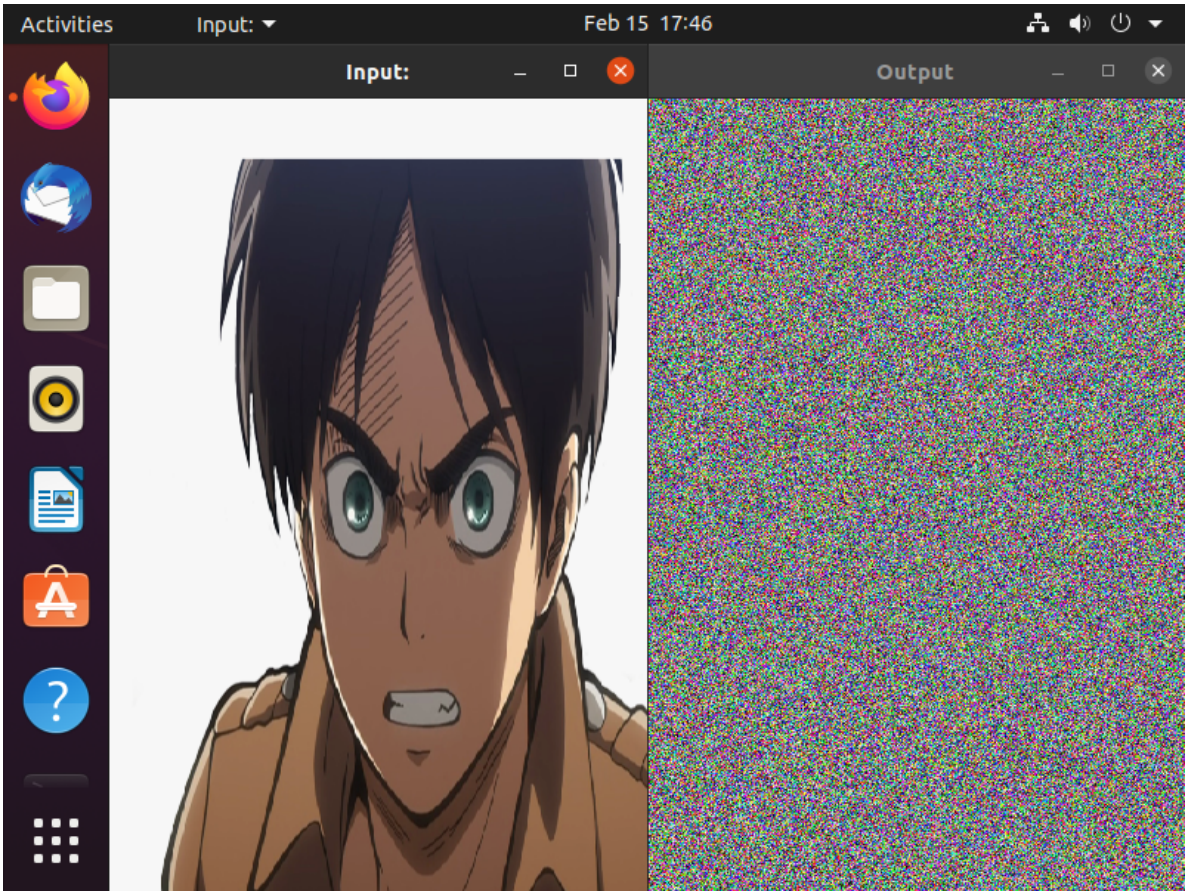
Furthermore, it was helpful learning about pixels and brushing up on how classes worked in C++ in a general sense. This assignment taught me a great deal about loading and creating new image files, as well.

In my main, I simply used the FibLFSR framework to pass an argument to the program– as you can see, it worked.

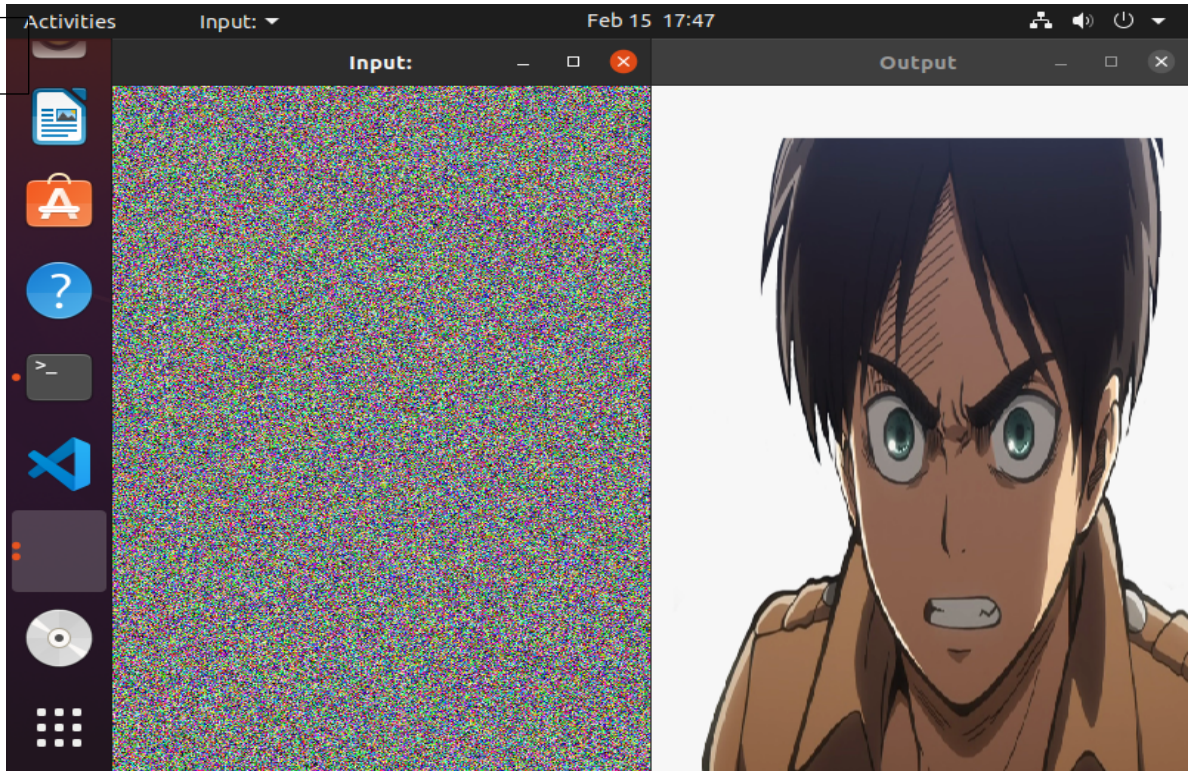


Command line, I ran the original picture and then ran its encoded version to ensure that everything was working as it should

My original picture, of the greatest anime character of all time



This was the original run, with my normal pic being encoded



This was the second one, as the prompt stated that if we did it correctly, the encoded picture should result in your original picture….

Makefile Feb 15 01:44:47 2021 1

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

PhotoMagic.cpp Mon Feb 15 17:51:52 2021 1

```
1: #include <iostream>
2: #include <string>
3:
4: #include <SFML/Graphics.hpp>
5:
6: #include "FibLFSR.h"
7:
8: using namespace std;
9:
10: void transform(sf::Image &image, FibLFSR *bit){ // transforms image using FibLFSR
11:
12:         int x, y; // variable declaration
13:
14:         int size1, size2; // for the x and y, respectively
15:
16:         sf::Sprite input, output; // self-explanatory input and output
17:
18:         sf::Texture texture_input, texture_output; // SFML textures
19:
20:         sf::Color p; // p for pixel
21:
22:         sf::Vector2u size = image.getSize();
```

```
23:
24:         sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Input:"); // input
25:
26:         texture_input.loadFromImage(image); // get the input image
27:
28:         input.setTexture(texture_input);   //  use the texture from the image
29:
30:         size1 = image.getSize().x; // get size, two variables for the x and y
31:
32:         size2 = image.getSize().y; // ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
33:
34:         for (x = 0; x < size1; x++){ // make the input negative
35:
36:                 for (y = 0; y < size2; y++){
37:
38:                         p = image.getPixel(x, y); // each pixel will be XOR'ed
39:
40:                         p.r = p.r ^ bit->generate(8);
41:
42:                         p.g = p.g ^ bit->generate(8);
43:
44:                         p.b = p.b ^ bit->generate(8);
45:
46:                         image.setPixel(x, y, p); // set it after the changes
47:                 }
48:         }
49:
50:         sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "Output"); // output
51:
52:         texture_output.loadFromImage(image);
53:
54:         output.setTexture(texture_output);
55:
56:         while (window1.isOpen() && window2.isOpen()){ // draw it to the output window
57:
58:                 sf::Event event;
59:
60:                 while (window1.pollEvent(event)){ if(event.type ==sf::Event::Closed)
window1.close(); } // close the windows if the event is done
```

```
61:
62:                 while (window2.pollEvent(event)){ if (event.type ==
sf::Event::Closed) window2.close(); } // ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
63:
64:                 window1.clear();
65:
66:                 window1.draw(input); // get to drawing
67:
68:                 window1.display();
69:
70:                 window2.clear();
71:
72:                 window2.draw(output);
73:
74:                 window2.display();
75:         }
76: }
77:
78: int main(int argc, char *argv[]){ // The main()  function  takes  three arguments
79:
80:     sf::Image image; // define an image for use
81:
82:         argv[0] = "./PhotoMagic"; // first argument will be the named executable
83:
84:         FibLFSR bit(argv[3]); // third argument will be the intial seed
85:
86:         if (!image.loadFromFile (argv[1])) exit(1); // if something goes wrong, exit
87:
88:         transform(image, &bit); // fingers crossed
89:
90:         if (!image.saveToFile(argv[2])) exit(1); // save the transformed file or exit
91:
92:         return 0;
93: }
94:
```

FibLFSR.h Sun Feb 07 23:10:12 2021 1

```
1: #ifndef FIBLFSR_H
2: #define FIBLFSR_H
```

```
3:
4: #include <iostream>
5: #include <string>
6: #include <vector>
7:
8: using namespace std;
9:
10: class FibLFSR {
11:
12:     public:
13:
14:     FibLFSR(string seed); // constructor to create LFSR
15:
16:     int step(); // simulate one step and return the new bit as 0 or 1
17:
18:     int generate(int k); // simulate k steps and return k-bit integer
19:
20:      ~FibLFSR(); // a destructor to deallocate memory from the constructor
21:
22:     friend std::ostream &operator<<(ostream &out, const FibLFSR &fibLFSR); //
overloads the << stream insertion operator to display its current value
23:
24:     private :
25:
26:     string str; // string for comparison
27:
28:     vector <int> memory; // for de-allocating memory at the end
29:
30:     int byte1, byte2, byte3, byte4; // variables to match the diagram
31: };
32:
33: #endif //FIBLFSR_H
```

FibLFSR.cpp Sun Feb 07 23:14:02 2021 1

```
1: #include <iostream>
2: #include "FibLFSR.h"
3:
4: using namespace std;
5:
```

```
6: FibLFSR::FibLFSR(string seed) { str = seed; } // initializes the seed
8: int FibLFSR::step(){ // function simulates one step of the LFSR
9:
10:     int compare;
11:     compare = 0;
12:
13:     byte3 = str.at(2) - '0';
14:     byte2 = str.at(3) - '0';
15:     byte1 = str.at(5) - '0';
16:     byte4 = str.at(0) - '0';
17:
18:     if ((byte4 == 1 && byte3 == 1) || (byte4 == 0 && byte3 == 0)){ compare = 0; }
19:     else{ compare = 1; }
20:     if (compare == byte2){ compare = 0; }
21:     else { compare = 1; }
22:     if (compare == byte1){ compare = 0; }
23:     else { compare = 1; }
24:
25:     str.erase(str.begin());        // make the bit spot at the beginning
open
26:     str.push_back(compare + '0');  // push back the output
27:     return compare; // return the result, which is the comparison
28: }
29:
30: int FibLFSR::generate(int k){
31:
32:     int i, j; // initialization variables
33:     j= 0;
34:
35:     for (i = 0; i < k; i++){
36:
37:
38:
39:     }
40:
41:     return j;
42: }
43:
44: FibLFSR:: ~FibLFSR(){ memory.clear(); } // deallocate any leftover memory
```

```
45:
46: ostream &operator <<(std::ostream &out, const FibLFSR &fibLFSR) {
47:       return out << fibLFSR.str;
48:   } // overload the << stream insertion operator
```

PS2: N-Body Simulation

PS2A:

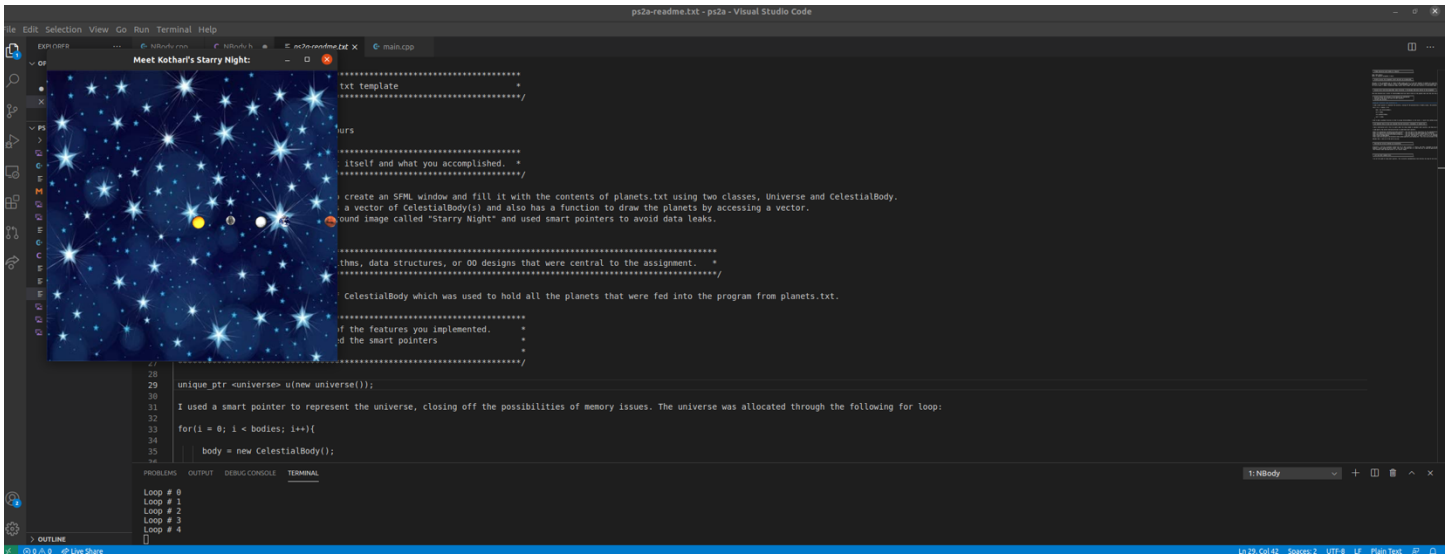
Purpose: To display a static solar system in an SFML window using coordinates that we would read in from a text file.

Key Parts: To correctly do this, the program was required to be able to read each line, saving each string into the corresponding variable. I did this using CelestialBody. This class was one of the most important parts of this assignment. Building upon that, the class Universe was a vector of CelestialBody which was used to hold all the planets that were fed into the program from planets.txt. Another central data structure to this assignment was a smart pointer, which we were learning about in class around the time of this assignment. I used a smart pointer by calling: unique_ptr <universe> u(new universe());

My Version: I used this smart pointer to represent the universe, closing off the possibilities of memory issues. The universe was allocated through a for loop. I had to make a pushback function in order to assign CelestialBody(s) to the vector, but I'd already had a basic understanding of how to do that, so it wasn't bad. For extra credit, I added a background image called "Starry Night" and used smart pointers to avoid data leaks.

What I Learned: Another great thing that this assignment taught me was the matter of perspective. I'd always wondered how you could accurately display a solar system, and this program taught me that manipulating the size of the window and multiplying position by scale and the aforementioned size would make it possible to display everything without a doubt.

This was the image I
used for the background…



Proof that my starry night
image worked itself into the
SFML window and that the
program was working correctly!

Makefile Mon Feb 22 19:40:13 2021 1

```
1: C= g++
2: CFLAGS= -Wall -Werror -ansi -pedantic
3: GFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
4: c11 = -std=c++11
5:
6: all:
7:         make NBody
8: NBody: NBody.o main.o
9:         g++ NBody.o main.o -o NBody $(GFLAGS) $(c11)
10: Nbody.o: NBody.cpp NBody.h
11:         g++ -c NBody.cpp -o NBody.o $(CFLAGS) $(c11)
12: main.o: main.cpp NBody.h
13:         g++ -c main.cpp -o main.o $(CFLAGS) $(c11)
14: clean:
15:         rm *.o *NBody
```

Main.cpp Mon Feb 22 22:04:15 2021 1

```
1: #include <string>
2: #include <iostream>
3:
4: #include "NBody.h"
5:
6: using namespace std;
7:
8: int main(int argc,char *argv[]){
9:
10:     int i, bodies; // variable declaration
11:     double radius;
12:
13:     CelestialBody *body;
14:
15:     sf::Image backgroundimage;
16:     sf::Texture background_texture;
17:     sf::Sprite background_sprite;
18:
19:     sf::RenderWindow window1(sf::VideoMode(500, 500), "Meet Kothari's Starry
Night:"); // load up an SFML window and call it Meet Kothari's Starry Night
20:
```

```
21:     backgroundimage.loadFromFile("starrynight.jpg"); // extra credit
22:
23:     background_texture.loadFromImage(backgroundimage); // load from the jpg
24:
25:     background_sprite.setTexture(background_texture); // set that texture
26:
27:     cout << scientific;
28:
29:     cin >> bodies >> radius;
30:
31:     window1.draw(background_sprite);
32:
33:     window1.display();
34:
35:     cout << "Input: " << bodies << " radius: " << radius << endl;
36:
37:     unique_ptr <universe> u (new universe()); // get ready to make the Universe
38:
39:     for(i = 0; i < bodies; i++){ // use the smart pointer, allocate memory
40:
41:       body = new CelestialBody();
42:
43:       cin >> *body;
44:
45:       (*u).pushback(*body);
46:
47:       cout << *body;
48:     }
49:
50:     (*u).draw(window1); // start drawing
51:
52:     window1.display(); // start displaying the drawings
53:
54:     while (window1.isOpen()){ // close the window once the drawing is done
55:
56:       sf::Event event;
57:
58:       while(window1.pollEvent(event)){
59:
```

```
60:         if (event.type == sf::Event::Closed) window1.close();
61:
62:         else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) window1.close();
63:
64:       }
65:
66:       window1.display(); // display out to the SFML window
67:     }
68:
69:     return 0;
70: }
```

NBody.h Mon Feb 22 22:02:15 2021 1

```
1: #ifndef _NBODY_H
2: #define _NBODY_H
3:
4: #include <iostream>
5: #include <math.h>
6: #include <stdlib.h>
7: #include <string>
8:
9:
10: #include <SFML/System.hpp>
11: #include <SFML/Window.hpp>
12: #include <SFML/Graphics.hpp>
13: #include <SFML/Audio.hpp>
14:
15: const sf::Vector2f window(500,500);
16: const double radius = 2.50e11;
17:
18: using namespace std;
19:
20: class CelestialBody: public sf::Drawable {
21:
22:     public:
23:
24:     CelestialBody(); // default
25:     CelestialBody(double x_position, double y_position, double x_velocity, double
y_velocity, double bmass, string planet); //  most important function, for positioning
```

```
26:
27:     void draw(sf::RenderTarget &obj, sf::RenderStates status) const; // draw!
28:
29:     friend ostream &operator <<(ostream &output, CelestialBody &c); // overload <<
30:     friend istream &operator >>(istream &input, CelestialBody &d);  // overload >>
31:
32:     private:
33:
34:     double mass; // for mass
35:
36:     double xpos; // for xpos
37:     double ypos; // for ypos
38:
39:     double xvel; // for xvel
40:     double yvel; // for yvel
41:
42:     string picture;
43:
44:     sf::Image pImage; // for the image
45:
46:     sf::Sprite pSprite; // for the sprites
47:
48:     sf::Texture pTexture; // for the textures
49: };
50:
51: class universe {
52:
53:     public:
54:
55:     void draw(sf::RenderWindow &window1){ // keep track of how many loops
56:
57:         for (unsigned int i = 0; i < uni.size();i++){
58:
59:             window1.draw(uni[i]);
60:
61:             cout << "Loop # " << i << endl;
62:         }
63:     }
64:
```

```
65:     void pushback(CelestialBody obj){ uni.push_back(obj); } // push back
66:
67:     private:
68:
69:     vector <CelestialBody> uni; // vector
70: };
71:
72: #endif
73:
```

NBody.cpp **Mon Feb 22 22:02:07** 2021 1

```
1: #include "NBody.h"
2:
3: CelestialBody::CelestialBody(){ return; } // default
4:
5: CelestialBody::CelestialBody(double x_position, double y_position, double x_velocity,
double y_velocity, double bmass, string planet){
6:
7:         sf::Sprite draw_plan; // get everything defined
8:
9:         xpos = x_position; // set the new positions according to the old ones
10:
11:         ypos = y_position; // ^^^^^^^^^^^^^^
12:
13:         xvel = x_velocity; // ^^^^^^^^^^^^^^
14:
15:         yvel = y_velocity; // ^^^^^^^^^^^^^^
16:
17:         mass = bmass; // ^^^^^^^^^^
18:
19:         picture = planet; // ^^^^^^^^^^
20:
21:         if(!pImage.loadFromFile(picture)){ return; } // return if it doesn't work
22:
23:         pTexture.loadFromImage(pImage); // load from the input(s) given
24:
25:         pSprite.setTexture(pTexture); // load from the texture(s) given
26:
27:         xpos = ((window.x/radius) * xpos/2) + (window.x / 2);   // sets value of axes
```

```
28:
29:         ypos = ((window.y/radius) * ypos/2) + (window.y / 2);  // ^^^^^^^
30:
31:         pSprite.setPosition(sf::Vector2f(xpos,ypos)); // set position
32:
33:         cout << xpos << endl << ypos << endl; // output the values
34: }
35:
36: void CelestialBody::draw(sf::RenderTarget &obj, sf::RenderStates status)
          const { obj.draw(pSprite); } // draw
37:
38: ostream &operator <<(ostream &output, CelestialBody &c){ // overlad output operator
39:
40:         output << "current x Position: " << c.xpos << endl; // output x position
41:
42:         output << "current y Postion: " << c.ypos << endl;  // output y position
43:
44:         output << "current x Velocity: " << c.xvel << endl; // output x velocity
45:
46:         output << "current y Velocity: " << c.yvel << endl; // output y velocity
47:
48:         output << "Particle Mass: " << c.mass << endl; // output particle mass
49:
50:         output << "Particle Name: " << c.picture << endl; // output particle name
51:
52:         return output; // return output
53: }
54:
55: istream &operator >>(istream &input, CelestialBody &d){ // overload input operator
56:
57:         input >> d.xpos >> d.ypos >> d.xvel >> d.yvel >> d.mass >> d.picture; // get
the positions, velocity, mass, and picture
58:
59:         if(!d.pImage.loadFromFile(d.picture)){ cout << "Error! Something went wrong
while loading the file..." << endl; } // in case something fails
60:
61:         d.pTexture.loadFromImage(d.pImage); // load the image from files
62:
63:         d.pSprite.setTexture(d.pTexture); // use the textures provided
```

```
64:
65:         d.xpos = ((window.x/radius) * d.xpos/2) + (window.x / 2); // set positioning
66:
67:         d.ypos = ((window.y/radius) * d.ypos/2) + (window.y / 2);
68:
69:         d.pSprite.setPosition(sf::Vector2f(d.xpos, d.ypos)); // setpos w values given
70:
71:         return input; // return the input
72:     }
73:
```

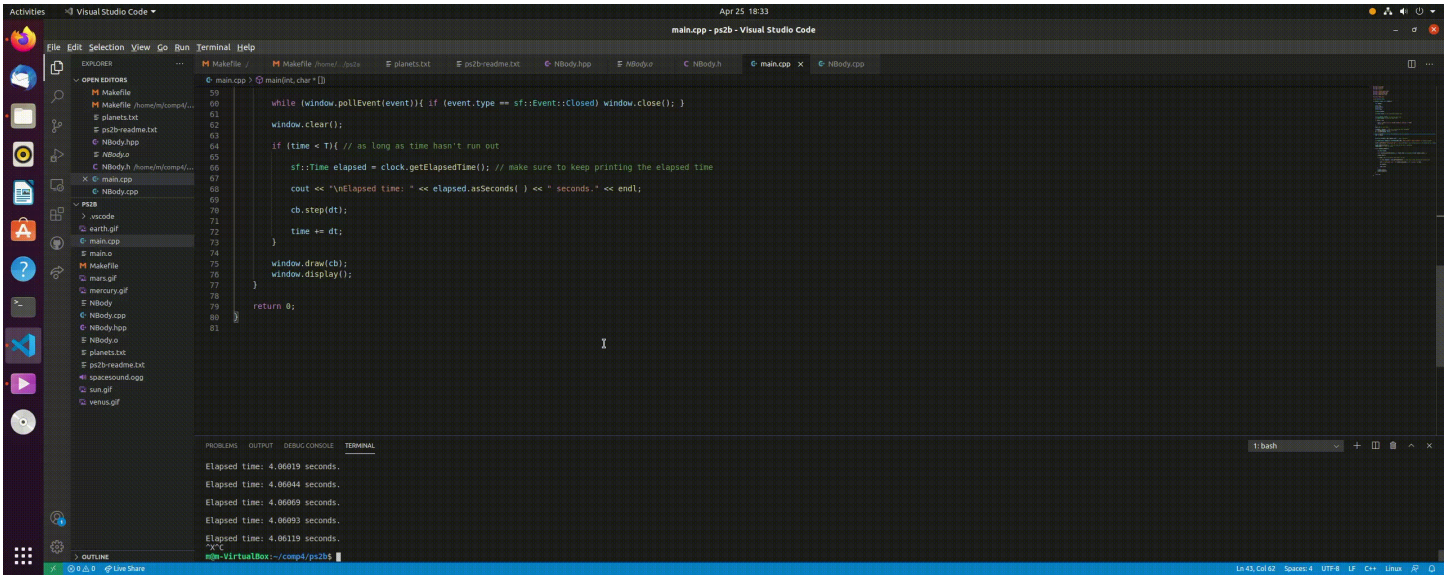
PS2B:

Purpose: To add physics simulation and animation to the program created in Part A.

Key Part: A very crucial part of this assignment was the implementation of a method named step in the class Universe which would take a time parameter (double seconds) and move the CelestialBody object given its internal velocity for that much time.

My Version: I actually didn't successfully complete this assignment. Well, I did, but I forgot to print the universe state, so that was a -1 for me. However, I did play a sound file when my simulation ran, so that negated any point deduction.

What I Learned: This assignment was very interesting to me. I learned (or re-learned, I suppose) a great deal about the physics that were involved with the N-Body diagram, and how to implement it- to be specific, Newton's Law of Universal Gravitation, and the leapfrog finite difference approximation. Another interesting thing was developing algorithms that I could see in action. It was really nice to be able to calculate the velocity and acceleration and watch them work in tandem with each other.



Makefile Wed Mar 03 12:57:19 2021 1

```
 1: C= g++
 2: CFLAGS= -Wall -Werror -ansi -pedantic
 3: GFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 4: c11 = -std=c++11
 5:
 6: all:
 7:         make NBody
 8: NBody: NBody.o main.o
 9:         g++ NBody.o main.o -o NBody $(GFLAGS) $(c11)
10: Nbody.o: NBody.cpp NBody.hpp
11:         g++ -c NBody.cpp -o NBody.o $(CFLAGS) $(c11)
12: main.o: main.cpp NBody.hpp
13:         g++ -c main.cpp -o main.o $(CFLAGS) $(c11)
14: clean:
15:         rm *.o *NBody
16:
```

Main.cpp Wed Mar 03 19:13:51 2021 1

```
1: #include <iostream>
2: #include <cstdlib>
3: #include <vector>
4:
5: #include <SFML/Graphics.hpp>
6: #include <SFML/Window.hpp>
7: #include <SFML/System.hpp>
8: #include <SFML/Audio.hpp>
9:
10: #include "NBody.hpp"
11:
12: using namespace std;
13:
14: int main(int argc, char* argv[]){
15:
16:     int cbodies;
17:
18:     double dt;
```

```
19:     double radius;
20:     double T;
21:     double time;
22:
23:     string filename;
24:
25:     sf::Clock clock; // for displaying elapsed time
26:
27:
28:     sf::SoundBuffer buffer; // load the audio file
29:     sf::Sound sound; // play the audio file
30:
31:     if (argc != 3){
32:
33:         cout << "\nThere are not enough arguments, exiting!" << endl;
34:         return -1;
35:     }
36:
37:     time = 0; // start time
38:
39:     filename = argv[0]; // make the filename the first arrgument
40:     T = strtod(argv[1], NULL); // max time
41:     dt = strtod(argv[2], NULL);
42:
43:     cin >> cbodies; // take in CelestialBodies and the radius
44:     cin >> radius;
45:
46:
47:     Universe cb(radius, 500, cbodies,cin); // make a universe
48:
49:     sf::RenderWindow window(sf::VideoMode(600, 600), "Meet Kothari's Solar System");
// display a window
50:
51:     buffer.loadFromFile("spacesound.ogg"); // load music
52:
53:         sound.setBuffer(buffer); // set the sound to the file I specified
54:         sound.play(); // play the sound
55:
56:     while (window.isOpen()){
```

```
57:
58:         sf::Event event;
59:
60:         while (window.pollEvent(event)){ if (event.type == sf::Event::Closed)
window.close(); }
61:
62:         window.clear();
63:
64:         if (time < T){ // as long as time hasn't run out
65:
66:             sf::Time elapsed = clock.getElapsedTime(); // print elapsed time
67:
68:             cout << "\nElapsed time: " << elapsed.asSeconds( ) << " seconds.";
70:             cb.step(dt);
71:
72:             time += dt;
73:         }
74:
75:         window.draw(cb);
76:         window.display();
77:     }
78:
79:         return 0;
80: }
81:
```

NBody.hpp Wed Mar 03 18:45:02 2021 1

```
1: #ifndef NBODY_HPP
2: #define NBODY_HPP
3:
4: #include <iostream>
5: #include <string>
6: #include <vector>
7: #include <memory>
8:
9: #include <SFML/Graphics.hpp>
10:
11: using namespace std;
12:
```

```
13:
14: class CelestialBody :public sf::Drawable {
15:
16:     private:
17:
18:     double winsize; // for the window
19:
20:     double xpos; // same as before
21:     double ypos;
22:
23:     double xvel;
24:     double yvel;
25:
26:     double mass;
27:     double radius;
28:
29:     double display_x;
30:     double display_y;
31:
32:     string filename;
33:
34:     sf::Sprite sprite;
35:     sf::Texture texture;
36:
37: public:
38:     //constructors
39:     CelestialBody(); // default
40:     CelestialBody(double x_pos, double y_pos, double x_vel, double y_vel,
 double m, string name, double radius, double winsize); // improved positioning
41:      ~CelestialBody(); // destructor
42:
43:     friend std::istream& operator >>(std::istream& input, CelestialBody&ci); //
istream
44:     friend std::ostream& operator <<(std::ostream& out, CelestialBody& co);   //
ostream
45:
46:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states)const; //
draw!
47:
```

```
48:     //accessor functions
49:     double get_posx(); // same as before
50:     double get_posy();
51:
52:     double get_velx();
53:     double get_vely();
54:
55:     double get_mass();
56:     string get_filename();
57:
58:     //mutators
59:     void set_x_y_pos(double x_input, double y_input); // newly added mutator
functions for everything
60:
61:     void set_velx(double vx); // velocity
62:     void set_vely(double vy); // ^^^^^^^^
63:
64:     void set_radius(double radius); // radius
65:     void set_window(double size); // window
66:
67:     void set_position(); // set pos
68:
69: };
70:
71: class Universe : public sf::Drawable { // newly defined Universe class, as per
comments on ps2a
72:
73:     public:
74:
75:     Universe(); // basic constructor
76:     Universe(double radius, int window, int num_of_planets, istream &in); // default
constructor
77:
78:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states)const; //
draw func for universe
79:
80:     friend ostream &operator <<(std::ostream& out, const Universe& co); // overlad <<
81:
```

```
82:     void step(double seconds); // take a time parameter (double seconds)and move the
CelestialBody object given its internal velocity for that much time
83:
84:     double get_r();
85:     int get_numPlanets();
86:
87:     void printInfo();
88:
89: private:
90:
91:     double r;
92:
93:     int numplanets;
94:     int winsize;
95:
96:     vector <std::unique_ptr <CelestialBody>> planets;
97:
98: };
99:
100: #endif //NBODY_HPP
```

NBody.cpp Wed Mar 03 19:10:53 2021 1

```
1: #include "NBody.hpp" // need the header file, of course
2:
3: #include <iostream>
4: #include <cmath>
5:
6:
7: using namespace std;
8:
9:
10: Universe::Universe(){ // basic constructor
11:
12:         r = 0; // initialize r to be 0
13:         winsize = 0; // winsize to be 0, cause it’s a default constructor
14: }
15:
16: Universe::Universe(double radius, int window, int num_of_planets, istream &in){
17:
```

```
18:         int i; // for the loop
19:
20:         r = radius; // set radius, window, numplanets to their updated values
21:         winsize = window; // update windowsize
22:         numplanets = num_of_planets; // update number of planets
23:
24:         for (i = 0; i < num_of_planets; i++){ // initialize planets
25:
26:                 unique_ptr <CelestialBody> ptr(new CelestialBody()); // smart pointer
27:
28:                 CelestialBody(); // assign a new CelestialBody
29:                 planets.push_back(move(ptr)); // push back
30:                 planets[i]->set_radius(r); // ith field to set radius
31:                 planets[i]->set_window(window); // set window
32:                 in >> *planets[i]; // take in planets[i]
33:         }
34:
35: }
36:
37: void Universe::draw(sf::RenderTarget &target, sf::RenderStates states) const { //
draw!
38:
39:         int i; // for loop
40:
41:         for (i = 0; i < numplanets; i++){ // as long as there are planets left...
42:
43:                 target.draw(*planets.at(i), states); // draw each of the planets
44: }
45: }
46:
47:
48: void Universe::step(double seconds){
49:
50:         int i, k; // initialization variables
51:
52:         double ax; // for each planet, implement net forces
53:         double ay;
54:
55:         double dx; // gonna need X and Y of each...
```

```
56:         double dy;
57:
58:         double force; // gravitational forces
59:         double forcex;
60:         double forcey;
61:
62:         double fx; // horizontal and vertical forces
63:         double fy;
64:
65:         double G; // for the gravitational constant
66:
66:
67:         double velx; // for velocity
68:         double vely;
69:
70:         double x2; // updated values of x and y
71:         double y2;
72:
73:         double r; // radius
74:
75:         for (i = 0; i < numplanets; i++){  // as long as there are planets left.
76:
77:                 fx = 0; // horizontal forces on a body, initialize at 0
78:                 fy = 0; // vertical forces on a body, initialize at 0
79:
80:                 for (k = 0; k < numplanets; k++){ // calculate total outside force
81:
82:                         if (k != i){
83:
84:                                 G = 6.67e-11; // gravitational constant
85:
86:                                 dx = planets[k]->get_posx() - planets[i]->get_posx();
87:                                 dy = planets[k]->get_posy() - planets[i]->get_posy();
88:
89:                                 r = sqrt(pow(dx, 2) + pow(dy, 2)); // use dx & dy
90:
91:                                 force = (G * planets[k]->get_mass() * planets[i]
>get_mass())/ pow(r, 2); // calculate forces
92:                                 forcex = force * (dx / r);
```

```
93:                                 forcey = force * (dy / r);
94:
95:                                 fy += forcey; // adding force to total net force
96:                                 fx += forcex;
97:
98:                         }
99:                 }
100:
101:                 ax = fx / planets[i]->get_mass(); // final values
102:                 ay = fy / planets[i]->get_mass();
103:
104:                 velx = planets[i]->get_velx() + seconds * ax;
105:                 vely = planets[i]->get_vely() + seconds * ay;
106:
107:                 planets[i]->set_velx(velx);
108:                 planets[i]->set_vely(vely);
109:
110:                 x2 = (planets[i]->get_posx()) + velx * seconds;
111:                 y2 = (planets[i]->get_posy()) + vely * seconds;
112:
113:                 planets[i]->set_x_y_pos(x2, y2);
114:         }
115: }
116:
117: void Universe::printInfo(){ // prints info out to screen
118:
119:         int i; // for loop
120:
121:         cout << numplanets << endl; // print out number of planets
122:         cout << r << endl; // print out radius
123:         for (i = 0; i < numplanets; i++){ // as long as there are planets left...
124:                 cout << planets[i]->get_posx() << " " << planets[i]->get_posy() << "
" << planets[i]->get_velx() << " " << planets[i]->get_vely() << " " << planets[i]-
>get_mass() << " " << planets[i]->get_filename() << endl;
125:         } // print out x pos, y pos, velx, vely, mass, and name
126: }
127:
128: double Universe::get_r(){ return r; } // return the radius value
129:
```

```
130: int Universe::get_numPlanets(){ return numplanets; } // return the number of planets
131:
132: CelestialBody::CelestialBody(){ // default constructor
133:
134:         winsize = 0;
135:         xpos = 0;
136:         ypos = 0;
137:         xvel = 0;
138:         yvel = 0;
139:         mass = 0;
140:         radius = 0;
141:         filename = "";
142: }
143:
144: CelestialBody::CelestialBody(double x_pos, double y_pos, double x_vel, double y_vel,
double m, string name, double rad, double window_size){
145:
146:         double radx;
147:         double rady;
148:
149:         xpos = x_pos; // updated values of xpos
150:         ypos = y_pos; // updated values of ypos
151:         xvel = x_vel; // updated values of xvel
152:         yvel = y_vel; // updated values of yvel
153:
154:         mass = m; // update mass
155:         radius = rad; // update radius
156:         winsize = window_size; // update window size
157:         filename = name; // update filename
158:
159:         radx = (winsize / 2) * (xpos / radius) + (winsize / 2); // update radius
160:         rady = (winsize / 2) * (ypos / radius) + (winsize / 2);
161:
162:
163:         texture.loadFromFile(filename);  // load texture from image
164:
165:
166:         sprite.setTexture(texture);
167:         sprite.setPosition(radx, rady);
```

```
168: }
169:
170: istream &operator >>(istream &in, CelestialBody &ci){
171:
172:         double radx;
173:         double rady;
174:
175:         in >> ci.xpos >> ci.ypos >> ci.xvel >> ci.yvel >> ci.mass >> ci.filename;
176:
177:         radx = (ci.winsize / 2) * (ci.xpos / ci.radius) + (ci.winsize / 2);
178:         rady = (ci.winsize / 2) * (ci.ypos / ci.radius) + (ci.winsize / 2);
179:
180:         ci.texture.loadFromFile(ci.filename);  // load texture from image
181:
182:         ci.sprite.setTexture(ci.texture); // set texture, position
183:         ci.sprite.setPosition(radx, rady);
184:
185:
186:         return in; // return input
187: }
188:
189: void CelestialBody::draw(sf::RenderTarget &target, sf::RenderStates states) const{
target.draw(sprite, states); } // draw
190:
191: CelestialBody::~CelestialBody(){} // destructor
192:
193: double CelestialBody::get_posx(){ return xpos; } // return the xpos
194: double CelestialBody::get_posy(){ return ypos; } // return the ypos
195: double CelestialBody::get_velx(){ return xvel; } // return the xvel
196: double CelestialBody::get_vely(){ return yvel; } // return the yvel
197: double CelestialBody::get_mass(){ return mass; } // return the mass
198:
199: string CelestialBody::get_filename(){ return filename; } // return the filename
200:
201: void CelestialBody::set_radius(double r){ radius = r; } // update with radius value
202: void CelestialBody::set_window(double size){ winsize = size; } // update the size
203: void CelestialBody::set_velx(double vx){ xvel = vx; } // update xvel
204: void CelestialBody::set_vely(double vy){ yvel = vy; } // update yvel
```

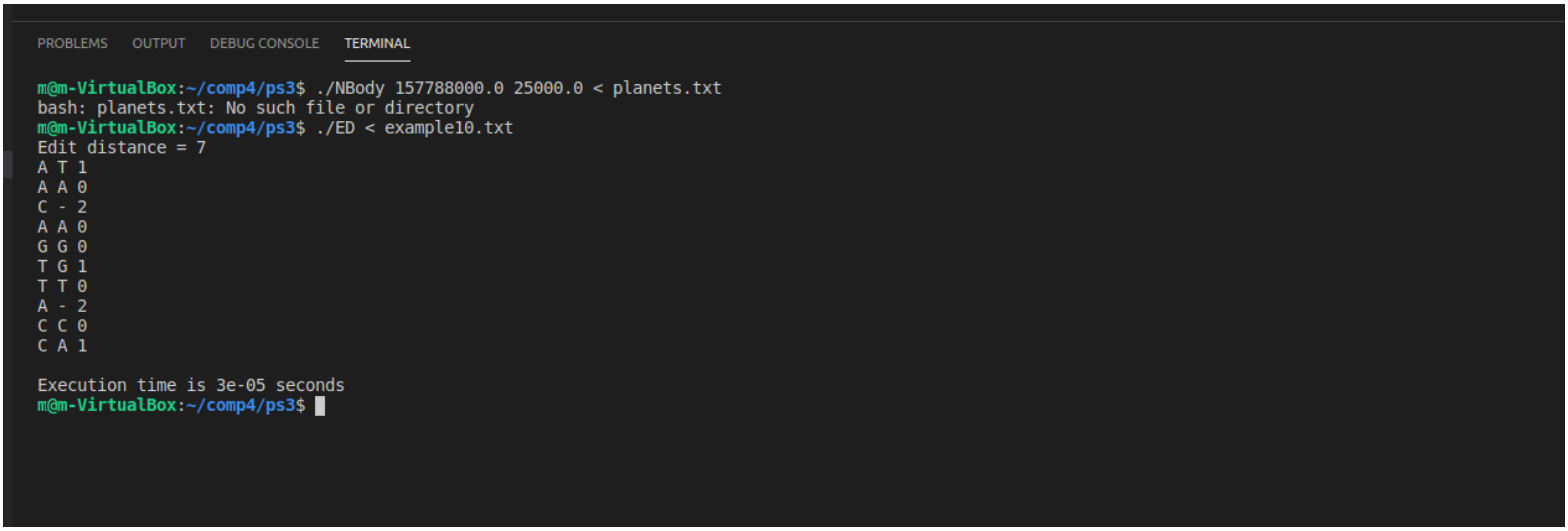
```
205: void CelestialBody::set_x_y_pos(double x_input, double y_input){ // feed the
positions as inputs to set new position
206:
207:         double radx; // declare variables for setting x_y pos
208:         double rady;
209:
210:         xpos = x_input; // xpos becomes your x_input
211:         ypos = y_input; // ypos becomes your y_input
212:
213:         radx = (winsize / 2) * (xpos / radius) + (winsize / 2); // get the xpos in
tandem with the radius
214:         rady = (winsize / 2) * (-ypos / radius) + (winsize / 2); // get the
ypos in tandem with the radius
215:
216:         sprite.setPosition(sf::Vector2f(radx, rady)); // set the position
217: }
218:
```

PS3: DNA Sequence Alignment

Purpose: The goal of this assignment was to solve a fundamental problem in computational biology and learn about a powerful programming paradigm known as dynamic programming.

Key Part: The real meat of the problem was learning how to calculate the edit distance of DNA sequence strings by matching the best possible alignment using the different options outlined in the prompt.

My Version/ What I Learned: I learned a great deal about the pros of using dynamic programming– some of which, especially in a prompt like the one that was given, are that you can use it to work on more complex programs and save results so that smaller portions of the program can reuse them to solve the original problem. This avoids recomputing the same quantity over and over again. The Needleman and Wunsch method (what I used) in particular, provides good time and memory complexity, and a multilinear time complexity. I learned a lot about complexity through this program, and learned the benefits of dynamic programming. Although it was similar to things I'd done in the past, it was also vastly different.



Makefile Mon Mar 15 13:33:06 2021 1

```
1: CC = g++
2: CFLAGS = -g -Wall -Werror -ansi -pedantic -std=c++11 -O3
3:
4: ED: ED.o main.o
5:         $(CC) $(CFLAGS) -o ED ED.o main.o -lsfml-system
6: all: ED
7: ED.o: ED.cpp ED.hpp
8:         $(CC) $(CFLAGS) -c ED.cpp
9: main.o: main.cpp
10:         $(CC) $(CFLAGS) -c main.cpp
11: clean:
12:          rm *.o ED
13:
```

Main.cpp Mon Mar 15 15:36:04 2021 1

```
1: #include <iostream>
2: #include <string>
3:
4: #include <SFML/System.hpp>
5:
6: #include "ED.hpp"
7:
8: using namespace std;
9:
10: int main(int argc, char *argv[]){ // two strings from stdin, use ED and then print
the result to stdout.
11:
12:     int opt_func; // describe variables
13:     string align_func; // will eventually call string Alignment()
14:
15:     string s1; // string 1 to be read
16:     string s2; // string 2 to be read
17:
18:     sf::Clock clock; // add code to calculate and print execution time
19:     sf::Time t;
20:
21:     cin >> s1; // pass in the two strings from stdin
22:     cin >> s2; // ^
```

```
23:
24:     ED ED(s1, s2); // make an ED
25:
26:     opt_func = ED.OptDistance();  // initialize two functions
27:     align_func = ED.Alignment(); // ^
28:
29:
30:     cout << "Edit distance = " << opt_func << ::endl; // output the ED
31:     cout << align_func << ::endl; // output result
32:
33:     t = clock.getElapsedTime(); // capture the running time
34:     cout << "Execution time is " << t.asSeconds() << " seconds \n"; // display time
35:
36:
37:     return 0; // end of main
38: }
39:
```

ED.hpp Mon Mar 15 15:34:45 2021 1

```
1: #ifndef ED_HPP
2: #define ED_HPP
3:
4: #include <iostream>
5: #include <vector>
6: #include <string>
7: #include <sstream>
8: #include <algorithm>
9:
10: #include <SFML/System.hpp>
11:
12: using namespace std;
13:
14: const int GAP = 2;
15:
16: class ED {
17:
18:     public:
19:
```

```
20:     ED(string a, string b); // a constructor that  accepts  the  two  strings  to  be
compared,  and  allocates  any  data  structures  necessary into order to do the work
21:
22:     static int penalty(char a, char b); // a static method int penalty(ch
ar a, char b) that returns the penalty for aligning chars and b (this will be a
0 or a 1).
23:
24:     static int min(int a, int b, int c); // returns  the  min  of  the  3 arguments.
25:
26:     int OptDistance(); // return the distance (from the [0][0] cell of the matrix
27:
28:     string Alignment(); // a method stringAlignment() return string w the alignment.
29:
30:      ~ED();
31:
32:     private:
33:
34:     string s1; // two strings for input, as per the prompt
35:
36:     string s2; // second string
37:
38:     vector < vector < int > > matrix; //
39: };
40:
41: #endif
42:
```

ED.cpp Mon Mar 15 16:39:11 2021 1

```
1: #include "ED.hpp"
2:
3: #include <string>
4: #include <vector>
5: #include <algorithm>
6:
7: using namespace std;
8:
9: ED::ED(string a, string b){ // a constructor that  accepts two  strings to compare
10:
11:     vector <int> temp;
```

```
12:
13:     s1 = a; // set values using the private members of the class  // temp to store
14:     s2 = b;
15:
16:     auto return_push = [&](int n) { temp.push_back(0); }; // lambda expression
17:
18:     for (unsigned j = 0; j < s2.length() + 1; j++){ return_push(0); }
19:     for (unsigned i = 0; i < s1.length() + 1; i++){ matrix.push_back(temp); }
20:
21: }
22:
23: ED::~ED(){} // destructor
24:
25: int ED::penalty(char a, char b){ // return the penalty for aligning chars and b
26:
27:     if (a == b) return 0; // if aligned, return 0
28:
29:     else return 1; // else return 1
30:
31: }
32:
33: int ED::min(int a, int b, int c){  // returns  the  minimum  of  the  3  arguments.
34:
35:   if (a <= b && a <= c) return a;
36:
37:   else if (b <= c) return b;
38:
39:   else return c;
40:
41: }
42:
43: int ED::OptDistance(){ // return distance (from the [0][0]cell of matrix when done).
44:
45:   int a, b, c, i, j; // a b and c will will be indices for the strings
46:   int str1, str2; // values to help initialzie the strings
47:
48:   i = 0; // initialize
49:   j = 0;
50:
```

```
51:   str1 = s1.length(); // initialize
52:   str2 = s2.length();
53:
54:   for (i = str1; i >= 0; i--) matrix[i][str2] = GAP * (str1 - i); // fill bottom row
and last column with gap penalty
55:   for (j = str2; j >= 0; j--) matrix[str1][j] = GAP * (str2 - j);
56:
57:   for (i = (str1 - 1); i >= 0; i--){ // fill the inside of the matrix
58:
59:     for (j = (str2 - 1); j >= 0; j--){
60:
61:       a = GAP + matrix[i + 1][j];
62:       b = GAP + matrix[i][j + 1];
63:       c = matrix[i + 1][j + 1] + penalty(s1[i], s2[j]);
64:
65:       matrix[i][j] = min(a, b, c);
66:
67:     }
68:
69:   }
70:
71:   return matrix[0][0];
72: }
73:
74: string ED::Alignment(){ // return a string that can be printed to display alignment.
75:
76:   stringstream return_string; // inherit from ostream functionality
77:
78:   int a, b, i, j; // indices for strings and other basic variables
79:   int str1, str2; // to access strings
80:   int position; // to represent the current position
81:   int penalty_result; // penalty result
82:
83:   i = 0; // initialize at 0
84:   j = 0;
85:
86:   str1 = s1.length(); // start accessing columns
87:   str2 = s2.length(); // start accessing rows
88:
```

```
89:   while (i < str1 || j < str2){ // as long as the matrix hasn’t been filled,
90:
91:     position = matrix[i][j]; // set position
92:
93:     a = i < str1 ? GAP + matrix[i + 1][j] : -1; // get to aligning
94:     b = j < str2 ? GAP + matrix[i][j + 1] : -1;
95:     penalty_result = penalty(s1[i], s2[j]);
96:
97:     if (position == a){ // in case we need to mve down
98:
99:       return_string << s1[i] << " - " <<  "2\n";
100:
101:       i++;
102:   }
103:
104:   else if (position == b){ // in case we need to move right
105:
106:     return_string << "- " << s2[j] << " 2\n";
107:
108:     j++;
109:
110:   }
111:
112:   else { // in case we need to move diagonal
113:
114:     return_string << s1[i] << " " << s2[j] << " " << penalty_result << " \n";
115:
116:     i++;
117:     j++;
118:
119:   }
120:
121:   }
122:
123:   string result_string = return_string.str();
124:
125:   return result_string;
126: }
127:
```

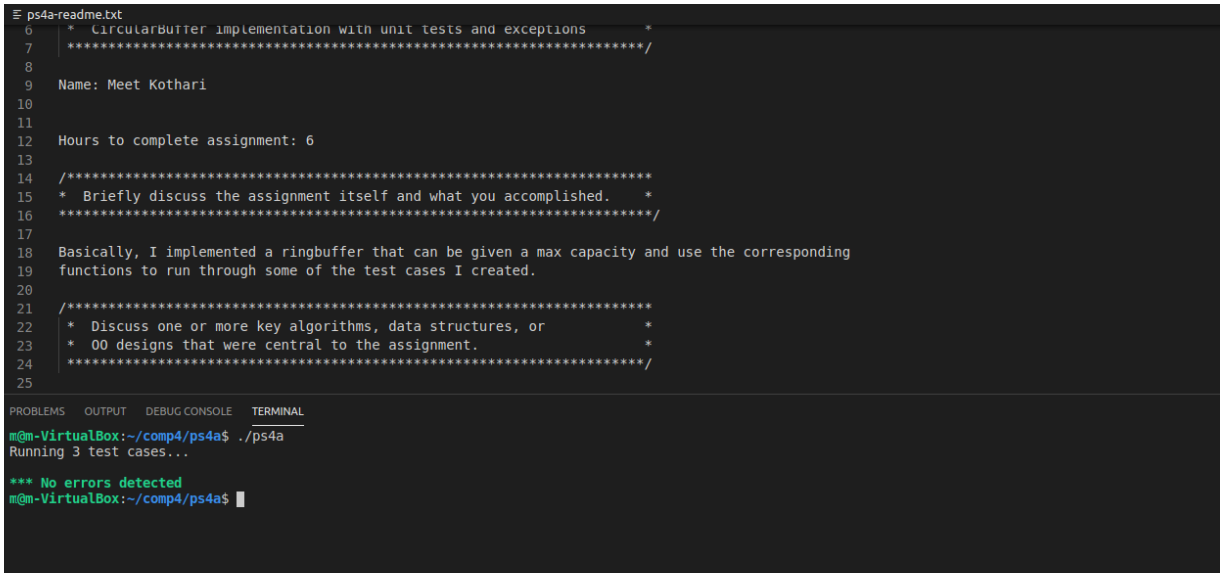
PS4: Synthesizing a Plucked String Sound

PS4A:

Purpose: The first part of this assignment had to do with the Karplus-Strong algorithm and laying groundwork. The two primary components that make the Karplus-Strong algorithm work are the ring buffer feedback mechanism and the averaging operation.

My Version: I did this by having a variable to hold the index of the first and last element in the buffer and then every time something was enqueued or dequeued, it looked to see if the next element to increment to was the last of the array or not and do the corresponding part.

Key Parts/Takeaways: I implemented the features as outlined by the prompt. The size, first, and last variables were incremented and decremented when enqueue and dequeue were called. These functions were important to the assignment because they were used to add and delete items. A major thing I learned in this assignment was how to use CPPLINT. While it was annoying to pick up and use, I can't deny how much cleaner it made my code look.



Makefile Mon Mar 22 21:14:41 2021 1

```
1: all: CircularBuffer ps4a
2:
3: CircularBuffer: main.o CircularBuffer.o
4:                 g++ -g main.o CircularBuffer.o -o CircularBuffer
5:
6: main.o: main.cpp
7:                 g++ -g -c main.cpp
8:
9: ps4a: test.o CircularBuffer.o
10:                 g++ -g test.o CircularBuffer.o -o ps4a -lboost_unit_test_framework
11:
12: test.o: test.cpp
13:                 g++ -g -c test.cpp -lboost_unit_test_framework
14:
15: CircularBuffer.o: CircularBuffer.cpp CircularBuffer.hpp
16:                 g++ -g -c CircularBuffer.cpp
17:
18: clean:
19:                 rm -rf *.o *.gch * ~ CircularBuffer ps4a
```

Main.cpp Mon Mar 22 21:14:58 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #include <iostream>
4: #include "CircularBuffer.hpp"
5:
6: int main(int argc, char *argv[]) {
7:     std::cout << "Running..." << std::endl;
8: }
9:
```

CircularBuffer.hpp Mon Mar 22 20:11:49 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #ifndef  _HOME_M_COMP4_PS4A_CIRCULARBUFFER_HPP_
4: #define  _HOME_M_COMP4_PS4A_CIRCULARBUFFER_HPP_
5:
6: #include <stdint.h>
```

```
7:
8: class CircularBuffer {
9:  public:
10:     explicit CircularBuffer(int capacity);  // create buffer with capacity
11:     int size();                 // return number of items in the buffer
12:     int capacity();             // return capacity
13:     bool isEmpty();             // is buffer empty (size equals zero)?
14:     bool isFull();              // is buffer full (size == capacity)
15:     void enqueue(int16_t x);    // add item x to the end
16:     int16_t dequeue();          // delete and return item from the front
17:     int16_t peek();             // return (but do not delete) item from front
18:     CircularBuffer();          // destructor
19:
20:  private:
21:     int16_t *buffer;               // buffer
22:     int _first, _last, _capacity, _size;  // first, last element
23: };
24:
25: #endif  // _HOME_M_COMP4_PS4A_CIRCULARBUFFER_HPP_
26:
```

CircularBuffer.cpp Mon Mar 22 20:49:15 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #include <stdint.h>
4: #include <iostream>
5: #include <exception>
6: #include <stdexcept>
7: #include "CircularBuffer.hpp"
8:
9: CircularBuffer::CircularBuffer(int capacity) {
10:     if (capacity < 1) {
11:         throw std::invalid_argument("_cb constructor: capacity must"
12:                                     " be greater than zero");
13:     }
14:     _size = 0;  // initialize
15:     _capacity = capacity;  // update value
16:     _last = capacity - 1;  // the last will be capacity-1
17:     _first = _last;  // new value
```

```
18:     buffer = new int16_t[_capacity];
19:
20:     // initialize the buffer to all 0's
21:     for (int x = 0; x < _capacity; x++) {
22:         buffer[x] = 0;
23:     }
24: }
25:
26: CircularBuffer::~CircularBuffer() {  // destructor
27:     delete [] buffer;
28: }
29:
30: int CircularBuffer::size() {
31:     return _size;  // pretty simple, update value
32: }
33:
34: int CircularBuffer::capacity() {
35:     return _capacity;
36: }
37:
38: bool CircularBuffer::isEmpty() {  // is the buffer empty (size equals zero)?
39:     if (size() == 0)  // return true if 0, else...
40:         return true;
41:     else
42:         return false;
43: }
44:
45: bool CircularBuffer::isFull() {  // is the buffer full  (size equals capacity)?
46:     if (size() == _capacity)  // return true if equal to capacity, else...
47:         return true;
48:     else
49:         return false;
50: }
51:
52: void CircularBuffer::enqueue(int16_t x) {  // add item x to the end
53:     // check if buffer is full before enqueuing
54:     if (isFull() == true) {
55:         throw std::runtime_error("enqueue: can't enqueue to a full"
56:                                  " buffer");
```

```
57:     }
58:     if (size() == 0) {  // if the buffer is empty...
59:         buffer[_last] = x;
60:     } else if (_last + 1 == _capacity) {  // if true loop to index 0
61:         _last = 0;
62:         buffer[_last] = x;
63:     } else {  // we can just loop last one normally
64:         _last += 1;
65:         buffer[_last] = x;
66:     }
67:     _size++;
68:     return;
69: }
70:
71: int16_t CircularBuffer::dequeue() {  // delete and return item from the front
72:     int16_t popped;
73:
74:     //  check if buffer is empty before dequeuing
75:     if (isEmpty() == true) {
76:         throw std::runtime_error("dequeue: can't dequeue an empty"
77:                                  " buffer");
78:     }
79:
80:     popped = buffer[_first];           // record the dequeued value
81:     buffer[_first] = ' ';             // cell is now empty
82:
83:     // determine how to increment first
84:     if (_first == _last)  {              // last element do nothing
85:         _size--;
86:         return popped;
87:     } else if (_first + 1 == _capacity) {  // loop to index 0
88:         _first = 0;
89:     } else {
90:         _first += 1;                   // incrememnt normally
91:     }
92:     _size--;
93:     return popped;
94: }
95:
```

```
96: int16_t CircularBuffer::peek() {
97:     //  check if buffer is empty before dequeuing
98:     if (isEmpty() == true) {
99:         throw std::runtime_error("peek: can't peek an empty"
100:                                  " ring");
101:     }
102:
103:     return buffer[_first];
104: }
```

Test.cpp Mon Mar 22 22:22:37 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #define BOOST_TEST_DYN_LINK
4: #define BOOST_TEST_MODULE Main
5: #include <boost/test/unit_test.hpp>
6: #include "RingBuffer.hpp"
7: #include <string>
8: #include <iostream>
9: #include <exception>
10: #include <stdexcept>
11:
12: BOOST_AUTO_TEST_CASE(RBcontructor) {
13:   // constructor
14:   BOOST_REQUIRE_NO_THROW(RingBuffer(100));
15:
16:   // should fail
17:   BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
18:   BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
19: }
20:
21: BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
22:   RingBuffer rb(100);
23:
24:   BOOST_REQUIRE(rb.isEmpty() == true);
25:
26:   rb.enqueue(2);
27:   rb.enqueue(1);
28:   rb.enqueue(0);
```

```
29:
30:   BOOST_REQUIRE(rb.size() == 3);
31:   BOOST_REQUIRE(rb.isFull() == false);
32:   BOOST_REQUIRE(rb.isEmpty() == false);
33:
34:   BOOST_REQUIRE(rb.dequeue() == 2);
35:   BOOST_REQUIRE(rb.dequeue() == 1);
36:   BOOST_REQUIRE(rb.dequeue() == 0);
37:
38:   // these 2 should fail
39:   BOOST_REQUIRE_THROW(rb.peek(), std::runtime_error);
40:   BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
41:
42:   BOOST_REQUIRE(rb.size() == 0);
43:
44:   BOOST_REQUIRE(rb.isFull() == false);
45:   BOOST_REQUIRE(rb.isEmpty() == true);
46: }
47:
48: BOOST_AUTO_TEST_CASE(RBfunctions) {
49:   RingBuffer rb2(3);
50:
51:   rb2.enqueue(2);
52:   rb2.enqueue(1);
53:   rb2.enqueue(0);
54:
55:   // buffer should be full
56:   BOOST_REQUIRE(rb2.size() == 3);
57:   BOOST_REQUIRE(rb2.isFull() == true);
58:   BOOST_REQUIRE(rb2.isEmpty() == false);
59:
60:   // should fail
61:   BOOST_REQUIRE_THROW(rb2.enqueue(3), std::runtime_error);
62: }
```

PS4B:

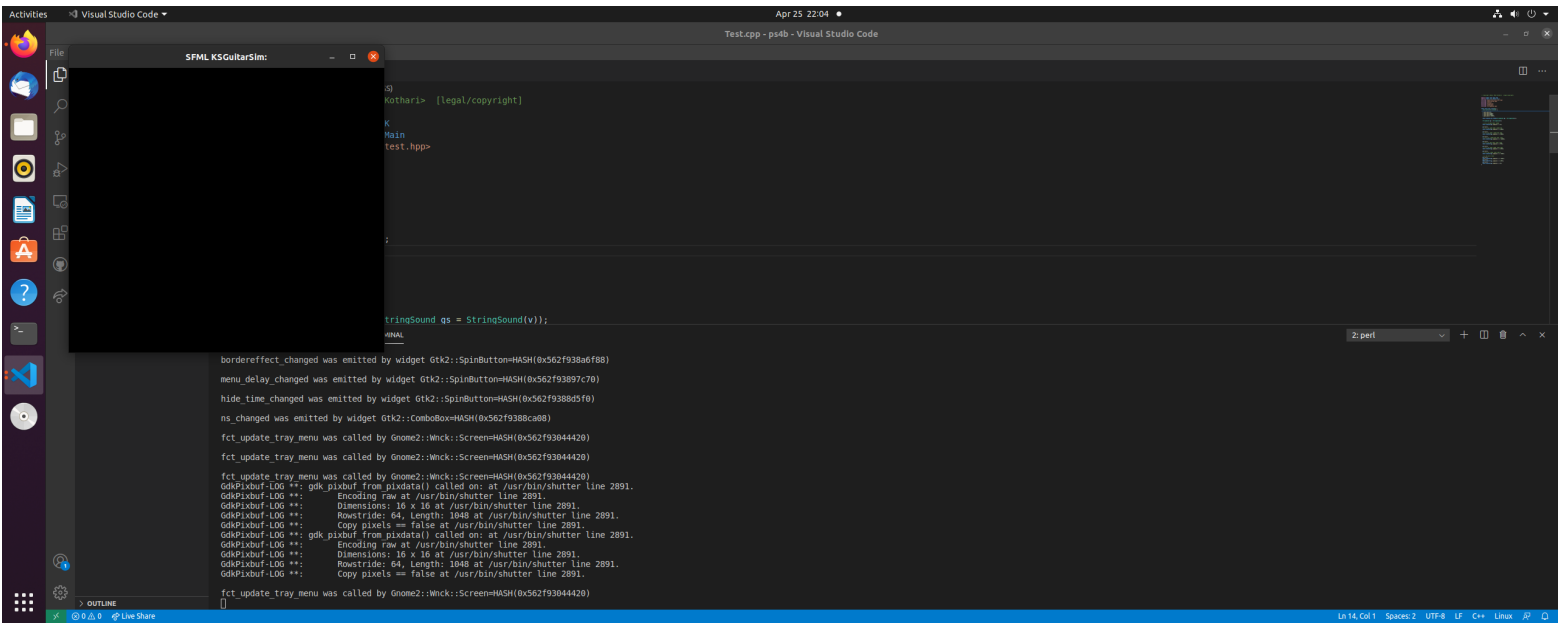
Purpose: This part of the assignment centered around implementing the Karplus-Strong guitar string simulation and generating a stream of string samples for audio playback under keyboard control.

My Version: I did this by creating a class StringSound, going to the private member variables declarations, and declaring a pointer to a CircularBuffer rather than declaring a CircularBuffer object itself.

Key Parts: A central part of this assignment was the pluck method, which I used to fill the guitar string's CircularBuffer with random numbers over the int16_t range. For the sound itself, I created StringSound and made the samples and sounds to be played for each key.

What I Learned: This assignment was very helpful with respect to SFML. I learned a lot about the sound functions and got a refresher in the windows and all, while also formatting my code to match CPPLINT.

I didn't do the extra credit so there's no image, just a screen that plays the sound



Makefile Mon Mar 29 21:11:34 2021 1

```
1: all: KSGuitarSim Test
2:
3: KSGuitarSim: KSGuitarSim.o CircularBuffer.o StringSound.o
4:                 g++ -Wall KSGuitarSim.o CircularBuffer.o StringSound.o -o KSGuitarSim
-lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window
5:
6: Test: Test.o StringSound.o CircularBuffer.o
7:                 g++ Test.o StringSound.o CircularBuffer.o -o Test -
lboost_unit_test_framework
8:
9: Test.o: Test.cpp
10:                 g++ -c Test.cpp -lboost_unit_test_framework
11:
12: KSGuitarSim.o: KSGuitarSim.cpp
13:                 g++ -Wall -c KSGuitarSim.cpp -lsfml-system -lsfml-audio -lsfml-
graphics -lsfml-window
14:
15: StringSound.o: StringSound.cpp StringSound.hpp
16:                 g++ -g -c StringSound.cpp
17:
18: CircularBuffer.o: CircularBuffer.cpp CircularBuffer.hpp
19:                 g++ -g -c CircularBuffer.cpp
20:
21: clean:
22:                 rm -rf *.o *.gch * ~ KSGuitarSim Test
```

KSGuitarsim.cpp Mon Mar 29 21:12:46 2021 1

```
1: /*
2:   Copyright 2015 Fred Martin,
3:   Y. Rykalova, 2020
4: */
5:
6: #include <math.h>
7: #include <limits.h>
8:
9: #include <iostream>
10: #include <string>
11: #include <exception>
```

```
12: #include <stdexcept>
13: #include <vector>
14:
15: #include <SFML/Graphics.hpp>
16: #include <SFML/System.hpp>
17: #include <SFML/Audio.hpp>
18: #include <SFML/Window.hpp>
19:
20: #include "CircularBuffer.hpp"
21: #include "StringSound.hpp"
22:
23: #define CONCERT_A 220.0
24: #define SAMPLES_PER_SEC 44100
25:
26: std::vector<sf::Int16> makeSamplesFromString(StringSound gs) {
27:   std::vector<sf::Int16> samples;
28:
29:   gs.pluck();
30:   int duration = 8;  // seconds
31:   int i;
32:   for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
33:     gs.tic();
34:     samples.push_back(gs.sample());
35:   }
36:
37:   return samples;
38: }
39:
40: int main() {
41:   sf::RenderWindow window(sf::VideoMode(300, 200), "SFML KSGuitarSim:");
42:   sf::Event event;
43:
44:   // Vector of all the sample vectors
45:   std::vector <std::vector <sf::Int16> > allsamples;
46:
47:   // Vector of 37 SoundBuffers
48:   std::vector <sf::SoundBuffer> SoundBuffers;
49:
50:   // Vector of 37 Sounds
```

```
51:   std::vector <sf::Sound> Sounds;
52:
53:   // Fill allsamples with 37 samples
54:   for (int x = 0; x < 37; x++) {
55:       std::vector<sf::Int16> samples;
56:       samples = makeSamplesFromString(StringSound(440 *
57:                                       (pow(2, ((x-24)/12)))));
58:       allsamples.push_back(samples);
59:   }
60:
61:   // Fill SoundBuffers with each SoundBuffer
62:   for (int x = 0; x < 37; x++) {
63:       sf::SoundBuffer buffer;
64:       if (!buffer.loadFromSamples(&allsamples[x][0],
65:           allsamples[x].size(), 2, SAMPLES_PER_SEC))
66:           throw std::runtime_error("sf::SoundBuffer: failed to "
67:                                    "load from samples.");
68:
69:       SoundBuffers.push_back(buffer);
70:   }
71:
72:   // Fill Sounds with all sound objects each conatining a SoundBuffer
73:   for (int x = 0; x < 37; x++) {
74:       sf::Sound sound;
75:       sound.setBuffer(SoundBuffers[x]);
76:       Sounds.push_back(sound);
77:   }
78:
79:   std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
80:   unsigned int i;
81:   while (window.isOpen()) {
82:     while (window.pollEvent(event)) {
83:       switch (event.type) {
84:        case sf::Event::Closed:
85:         window.close();
86:         break;
87:
88:        case sf::Event::TextEntered:
89:         i = keyboard.find(event.text.unicode);
```

```
90:         if (i < 37)
91:           Sounds[i].play();
92:         break;
93:        default:
94:       break;
95:       }
96:     }
97:
98:     window.clear();
99:     window.display();
100:   }
101:   return 0;
102: }
103:
```

CircularBuffer.hpp Mon Mar 29 20:11:49 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #ifndef  _HOME_M_COMP4_PS4B_CIRCULARBUFFER_HPP_
4: #define  _HOME_M_COMP4_PS4B_CIRCULARBUFFER_HPP_
5:
6: #include <stdint.h>
7:
8: class CircularBuffer {
9:  public:
10:     explicit CircularBuffer(int capacity);  // create buffer with capacity
11:     int size();                 // return number of items in the buffer
12:     int capacity();             // return capacity
13:     bool isEmpty();             // is buffer empty (size equals zero)?
14:     bool isFull();              // is buffer full (size == capacity)
15:     void enqueue(int16_t x);    // add item x to the end
16:     int16_t dequeue();          // delete and return item from the front
17:     int16_t peek();             // return (but do not delete) item from front
18:     CircularBuffer();          // destructor
19:
20:  private:
21:     int16_t *buffer;               // buffer
22:     int _first, _last, _capacity, _size;  // first, last element
23: };
```

```
24:
25: #endif  // _HOME_M_COMP4_PS4B_CIRCULARBUFFER_HPP_
26:
```

CircularBuffer.cpp Mon Mar 22 20:49:15 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #include <stdint.h>
4: #include <iostream>
5: #include <exception>
6: #include <stdexcept>
7: #include "CircularBuffer.hpp"
8:
9: CircularBuffer::CircularBuffer(int capacity) {
10:     if (capacity < 1) {
11:         throw std::invalid_argument("_cb constructor: capacity must"
12:                                     " be greater than zero");
13:     }
14:     _size = 0;  // initialize
15:     _capacity = capacity;  // update value
16:     _last = capacity - 1;  // the last will be capacity-1
17:     _first = _last;  // new value
18:     buffer = new int16_t[_capacity];
19:
20:     // initialize the buffer to all 0's
21:     for (int x = 0; x < _capacity; x++) {
22:         buffer[x] = 0;
23:     }
24: }
25:
26: CircularBuffer::~CircularBuffer() {  // destructor
27:     delete [] buffer;
28: }
29:
30: int CircularBuffer::size() {
31:     return _size;  // pretty simple, update value
32: }
33:
34: int CircularBuffer::capacity() {
```

```
35:     return _capacity;
36: }
37:
38: bool CircularBuffer::isEmpty() {  // is the buffer empty (size equals zero)?
39:     if (size() == 0)  // return true if 0, else...
40:         return true;
41:     else
42:         return false;
43: }
44:
45: bool CircularBuffer::isFull() {  // is the buffer full  (size equals capacity)?
46:     if (size() == _capacity)  // return true if equal to capacity, else...
47:         return true;
48:     else
49:         return false;
50: }
51:
52: void CircularBuffer::enqueue(int16_t x) {  // add item x to the end
53:     // check if buffer is full before enqueuing
54:     if (isFull() == true) {
55:         throw std::runtime_error("enqueue: can't enqueue to a full"
56:                                  " buffer");
57:     }
58:     if (size() == 0) {  // if the buffer is empty...
59:         buffer[_last] = x;
60:     } else if (_last + 1 == _capacity) {  // if true loop to index 0
61:         _last = 0;
62:         buffer[_last] = x;
63:     } else {  // we can just loop last one normally
64:         _last += 1;
65:         buffer[_last] = x;
66:     }
67:     _size++;
68:     return;
69: }
70:
71: int16_t CircularBuffer::dequeue() {  // delete and return item from the front
72:     int16_t popped;
73:
```

```
74:     //  check if buffer is empty before dequeuing
75:     if (isEmpty() == true) {
76:         throw std::runtime_error("dequeue: can't dequeue an empty"
77:                                  " buffer");
78:     }
79:
80:     popped = buffer[_first];           // record the dequeued value
81:     buffer[_first] = ' ';             // cell is now empty
82:
83:     // determine how to increment first
84:     if (_first == _last)  {              // last element do nothing
85:         _size--;
86:         return popped;
87:     } else if (_first + 1 == _capacity) {  // loop to index 0
88:         _first = 0;
89:     } else {
90:         _first += 1;                   // incrememnt normally
91:     }
92:     _size--;
93:     return popped;
94: }
95:
96: int16_t CircularBuffer::peek() {
97:     //  check if buffer is empty before dequeuing
98:     if (isEmpty() == true) {
99:         throw std::runtime_error("peek: can't peek an empty"
100:                                  " ring");
101:     }
102:
103:     return buffer[_first];
104: }
```

StringSound.hpp Mon Mar 29 21:12:58 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #include <vector>
4: #include "CircularBuffer.hpp"
5:
6: #ifndef _HOME_M_COMP4_PS4B_STRINGSOUND_HPP_
```

```
7: #define _HOME_M_COMP4_PS4B_STRINGSOUND_HPP_
8:
9: class StringSound {
10: public:
11:     explicit StringSound(double frequency);
12:     // create string
13:     // using a sampling rate of 44,100
14:     explicit StringSound(std::vector<sf::Int16> init);
15:     // create a guitar string with
16:     // size and initial values are given by
17:     // the vect
18:     StringSound(const StringSound &obj) {}  // no copy const
19:      ~StringSound();  // destructor
20:     void pluck();  // pluck the guitar string by replacing
21:     // the buffer with random values,
22:     // representing white noise
23:     void tic();  // advance the simulation one time step
24:     sf::Int16 sample();  // return the current sample
25:     int time();  // return number of times tic was called
26: // so far
27:
28:  private:
29:     CircularBuffer *_cb;
30:     int _time;
31: };
32: #endif  // _HOME_M_COMP4_PS4B_STRINGSOUND_HPP_
```

StringSound.cpp Mon Mar 29 20:44:04 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #include <math.h>
4: #include <stdint.h>
5: #include <vector>
6: #include <SFML/Audio.hpp>
7:
8: #include "StringSound.hpp"
9:
10: StringSound::StringSound(double frequency) {
11:     // create a guitar string sound of the
```

```
12:     // given frequency using a sampling rate
13:     // of 4410
14:     int size;
15:     _time = 0;
16:     size = ceil(44100/frequency);
17:     _cb = new CircularBuffer(size);
18:     pluck();
19: }
20: StringSound::StringSound(std::vector<sf::Int16> init) {
21:      // create a guitar string with
22:      // size and initial values are given by
23:      // the vector
24:     int i;
25:     _cb = new CircularBuffer(init.size());
26:     for (i = 0; i < init.size(); i++) {
27:         _cb->enqueue(init[i]);
28:     }
29: }
30: StringSound::~StringSound() {
31:     // delete _cb
32:     delete _cb;
33: }
34: void StringSound::pluck() {
35:     // pluck the guitar string by replacing
36:     // the buffer with random values,
37:     // representing white noise
38:     int random, newcap;
39:     newcap = _cb->capacity();
40:     _cb = new CircularBuffer(newcap);
41:     for (int x = 0; x < newcap; x++) {
42:         _cb->enqueue( (sf::Int16)(rand() & 0xffff));  //NOLINT
43:     }
44: }
45: void StringSound::tic() {
46:     // advance the simulation one time step
47:     int16_t popped = _cb->dequeue();
48:     int16_t peek = _cb->peek();
49:     popped = .996*((popped + peek)/2);
50:     _cb->enqueue(popped);
```

```
51:     _time++;
52: }
53: sf::Int16 StringSound::sample() {
54:     // return the current sample
55:     return _cb->peek();
56: }
57: int StringSound::time() {
58:     // return number of times tic was called
59:     // so far
60:     return _time;
61: }
62:
```

Test.cpp Mon Mar 29 21:09:16 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #define BOOST_TEST_DYN_LINK
4: #define BOOST_TEST_MODULE Main
5: #include <boost/test/unit_test.hpp>
6: #include <SFML/System.hpp>
7: #include <vector>
8: #include <exception>
9: #include <stdexcept>
10: #include "StringSound.hpp"
11:
12: BOOST_AUTO_TEST_CASE(GS) {
13:   std::vector<sf::Int16> v;
14:
15:   v.push_back(0);
16:   v.push_back(2000);
17:   v.push_back(4000);
18:   v.push_back(-10000);
19:
20:   BOOST_REQUIRE_NO_THROW(StringSound gs = StringSound(v));
21:
22:   StringSound gs = StringSound(v);
23:
24:   // GS is 0 2000 4000 -10000
25:   BOOST_REQUIRE(gs.sample() == 0);
```

```
26:
27:   gs.tic();
28:   // it's now 2000 4000 -10000 996
29:   BOOST_REQUIRE(gs.sample() == 2000);
30:
31:   gs.tic();
32:   // it's now 4000 -10000 996 2988
33:   BOOST_REQUIRE(gs.sample() == 4000);
34:
35:   gs.tic();
36:   // it's now -10000 996 2988 -2988
37:   BOOST_REQUIRE(gs.sample() == -10000);
38:
39:   gs.tic();
40:   // it's now 996 2988 -2988 -4483
41:   BOOST_REQUIRE(gs.sample() == 996);
42:
43:   gs.tic();
44:   // it's now 2988 -2988 -4483 1984
45:   BOOST_REQUIRE(gs.sample() == 2988);
46:
47:   gs.tic();
48:   // it's now -2988 -4483 1984 0
49:   BOOST_REQUIRE(gs.sample() == -2988);
50:
51:   // a few more times
52:   gs.tic();
53:   BOOST_REQUIRE(gs.sample() == -4483);
54:   gs.tic();
55:   BOOST_REQUIRE(gs.sample() == 1984);
56:   gs.tic();
57:   BOOST_REQUIRE(gs.sample() == 0);
58: }
59:
```

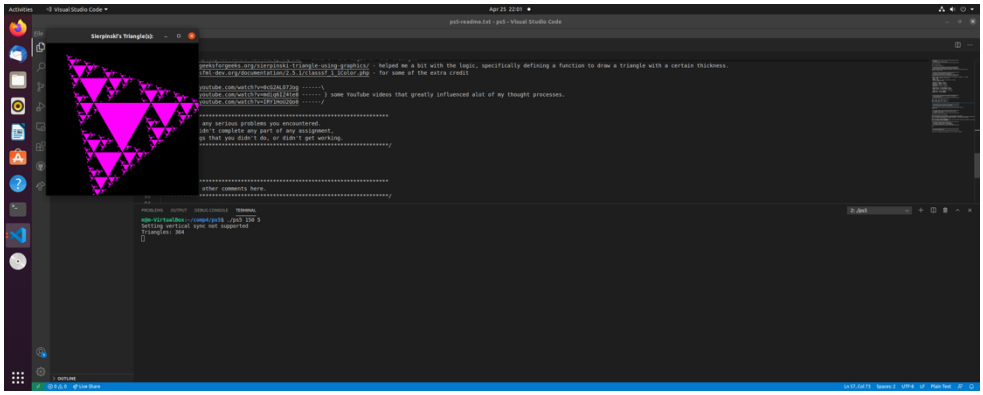
PS5: Recursive Graphics

Purpose: In this assignment, I created a program that takes command-line arguments L and N, uses recursive function fTree() and plots a Triangle Fractal. Specifically, a variation of the Sierpinski triangle.

My Version: For the triangle itself, I used an approach that calculated the center coordinates for the three corner triangles on each layer and recursively built them, saving them into a container to be drawn.

Key Parts: I'd say the most key concept in this assignment was recursion. The first step was to understand the abstraction that the Triangle class would provide for designing the recursive solution. Having decided the manner in which to attack the problem, after that, I just created them recursively by calculating their relative positions. It's what the whole assignment centered around.

What I Learned: My main takeaways from this assignment were how to draw shapes using the SFML library, and how to use advanced recursion. Prior to this assignment, I didn't have the most experience in using recursion, but I found it wholly beneficial to use an advanced application of recursion such as this and learn along the way. I made mistakes, but this assignment still aided me in finding out how to go about using that recursion. I also got some experience in the SFML color libraries.



I did the extra credit so mine was pink!

Makefile Tue Apr 13 00:17:42 2021 1

```
1: CC = g++
2: CFLAGS = -Wall -Werror -pedantic -g -lsfml-audio -lsfml-graphics -lsfml-window -lsfml-
system
3: OBJS = TFractal.o Triangle.o
4: DEPS = Triangle.hpp
5: LIBS =
6: EXE = ps5
7:
8: all: $(OBJS)
9:         $(CC) $(OBJS) -o $(EXE) $(LIBS) $(CFLAGS)
10:
11: TFractal.o: TFractal.cpp Triangle.hpp
12:         $(CC) $(CFLAGS) -c TFractal.cpp
13:
14: Triangle.o: Triangle.cpp Triangle.hpp
15:         $(CC) $(CFLAGS) -c Triangle.cpp
16:
17: clean:
18:         rm $(OBJS) $(EXE)
```

TFractal.cpp Tue Apr 13 01:41:09 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: // write a program TFractal.cpp with a recursive function fTree(),
4: // and a main() program that calls the recursive function.
5:
6: #include <iostream>
7: #include <cstdlib>
8: #include "Triangle.hpp"
9:
10: void fTree(Triangle &tri, double iterations, std::vector<Triangle> &trianglevect) {
11:   if (iterations == 0) return;
12:   Triangle tri2(tri.getsize()/2.0, tri.getcentx()-((1.0/2.0)*(tri.getsize())),
tri.getcenty()-((2.0/3.0)*(tri.getheight())));  // NOLINT
13:   trianglevect.push_back(tri2);
14:   fTree(tri2, iterations-1, trianglevect);
15:   Triangle tri3(tri.getsize()/2.0, tri.getcentx()+((3.0/4.0)*(tri.getsize())),
tri.getcenty()-((1.0/6.0)*(tri.getheight())));  // NOLINT
```

```
16:   trianglevect.push_back(tri3);
17:   fTree(tri3, iterations-1, trianglevect);
18:   Triangle tri4(tri.getsize()/2.0, tri.getcentx()-((1.0/4.0)*(tri.getsize())),
tri.getcenty()+((5.0/6.0)*(tri.getheight())));  // NOLINT
19:   trianglevect.push_back(tri4);
20:   fTree(tri4, iterations-1, trianglevect);
21:
22:   return;
23: }
24: int main(int argc, char *argv[]) {
25:   double L, N;
26:   // your program shall take two command-line arguments L and N:
27:   std::vector<Triangle> trianglevect;
28:   sf::Event event;
29:
30:   L = atoi(argv[1]);
31:   N = atoi(argv[2]);
32:
33:   Triangle tri(L, 200, 200);
34:
35:   trianglevect.push_back(tri);
36:   fTree(tri, N, trianglevect);
37:
38:   sf::RenderWindow window(sf::VideoMode(400, 400), "Sierpinski's Triangle(s):"); 39:
40:   cout << "Triangles: " << trianglevect.size() << endl;
41:
42:   while (window.isOpen()) {
43:     while (window.pollEvent(event)) {
42:   while (window.isOpen()) {
43:     while (window.pollEvent(event)) {
44:       if (event.type == sf::Event::Closed) {
45:         window.close();
46:       }
47:       window.clear();
48:       for (auto x : trianglevect) {
49:         window.draw(x);
50:       }
51:       window.display();
52:     }
```

```
53:   }
54:   return 0;
55: }
56:
```

Triangle.hpp Tue Apr 13 01:31:13 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #ifndef _HOME_M_COMP4_PS5_TRIANGLE_HPP_
4: #define _HOME_M_COMP4_PS5_TRIANGLE_HPP_
5:
6: #include <stdio.h>
7: #include <iostream>
8: #include <string>
9: #include <fstream>
10: #include <vector>
11: #include <memory>
12: #include <SFML/System.hpp>
13: #include <SFML/Window.hpp>
14: #include <SFML/Graphics.hpp>
15:
16: using namespace std; // NOLINT
17:
18: class Triangle: public sf::Drawable {
19:  public:
20:   // create a Triangleclass that derives from sf::Drawable.
21:   Triangle(double size, double centx, double centy);
22:
23:   double getsize() const;  // get the size
24:   double getcentx() const;  // center x
25:   double getcenty() const;  // center y
26:   double getheight() const;  // return heigh
27:
28:   double getx() const;  // return x
29:   double gety() const;  // return y
30:   double getx2() const;  // return x2
31:   double gety2() const;  // return y2
32:   double getx3() const;  // returen x3
33:   double gety3() const;  // return y3
```

```
34:
35:   void setsize(double val);  // set the size
36:   void setcentx(double val);  // setter for x center point
37:   void setcenty(double val);  // setter for y center point
38:   void setheight(double val);  // set height
39:
40:   void setx(double val);  // set x
41:   void sety(double val);  // set y
42:   void setx2(double val);  // set x2
43:   void sety2(double val);  // set y2
44:   void setx3(double val);  // set x3
45:   void sety3(double val);  // set y3
46:
47:  private:
48:   double newsize, newheight;
49:   double centerX, centerY;
50:   double x, y, x2, y2, x3, y3;
51:   virtual void draw(sf::RenderTarget &target, sf::RenderStates states) const;
52:   sf::VertexArray shape;
53: };
54:
55: void fTree(Triangle &tri, double iterations, std::vector<Triangle>
56:
57: #endif  // _HOME_M_COMP4_PS5_TRIANGLE_HPP_
```

Triangle.cpp Tue Apr 13 01:53:12 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #include <math.h>
4: #include "Triangle.hpp"
5:
6: using namespace sf;  // NOLINT
7:
8: Triangle::Triangle(double size, double centx, double centy):shape(sf::Triangles, 3) {
9:   newsize = size;  // update all of the values
10:   centerX = centx;
11:   centerY = centy;
12:   newheight = static_cast <double> ((sqrt(3.0)/2.0)) * newsize;
13:   // it's a triangle, after all
```

```
14:
15:   setx(centerX - newsize * (1.0/2.0));
16:   sety(centerY - newheight * (1.0/3.0));
17:   setx2(centerX + newsize * (1.0/2.0));
18:   sety2(centerY - newheight * (1.0/3.0));
19:   setx3(centerX);
20:   sety3(centerY + ((2.0/3.0) * newheight));
21:
22:   shape[0].position = sf::Vector2f(getx(), gety());
23:   shape[1].position = sf::Vector2f(getx2(), gety2());
24:   shape[2].position = sf::Vector2f(getx3(), gety3());
25:
26:   shape[0].color = sf::Color::Magenta;  // extra credit
27:   shape[1].color = sf::Color::Magenta;
28:   shape[2].color = sf::Color::Magenta;
29: }
30: double Triangle::getsize() const { return newsize; }  // pretty self-explanatory
31: double Triangle::getheight() const { return newheight; }  // update the values
32: double Triangle::getcentx() const { return centerX; }
33: double Triangle::getcenty() const { return centerY; }
34:
35: double Triangle::getx() const { return x; }
36: double Triangle::gety() const { return y; }
37:
38: double Triangle::getx2() const { return x2; }
39: double Triangle::gety2() const { return y2; }
40: double Triangle::getx3() const { return x3; }
41: double Triangle::gety3() const { return y3; }
42:
43: void Triangle::setsize(double val) { newsize = val; }
44: void Triangle::setheight(double val) { newheight = val; }
45: void Triangle::setcentx(double val) { centerX = val; }
46: void Triangle::setcenty(double val) { centerY = val; }
47:
48: void Triangle::setx(double val) { x = val; }
49: void Triangle::sety(double val) { y = val; }
50:
51: void Triangle::setx2(double val) { x2 = val; }
52: void Triangle::sety2(double val) { y2 = val; }
```

```
53: void Triangle::setx3(double val) { x3 = val; }
54: void Triangle::sety3(double val) { y3 = val; }
55:
56: void Triangle::draw(sf::RenderTarget &target, sf::RenderStates states) const {
57:   // you can have it justdrawitself to your main window
58:   target.draw(shape, states);
59:   return;
60: }
61:
```

PS6: Kronos Time Clock: Introduction to Regular Expression Parsing

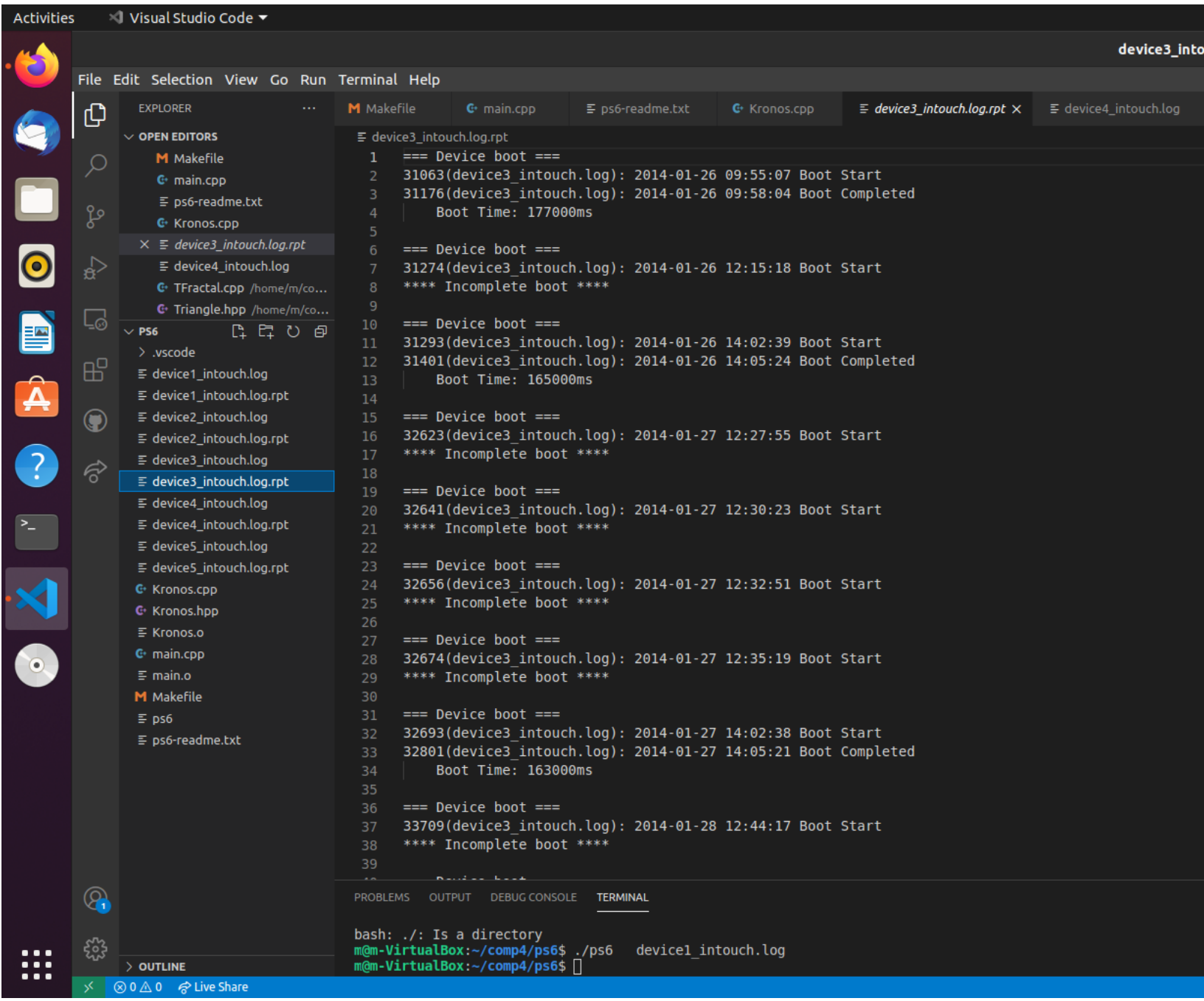
Purpose: Using regular expressions to parse log files generated from Kronos source files to evaluate the behavior of device's booting sequence.

My Version/ Key Parts: This assignment allowed for a bit of individualism. My approach relied on regex. It used regex to determine when the server was booted and then saved the date, when it was started, and time to determine how long the server was active.

I used some different ones. Mostly, I just matched things like the date and time.

What I Learned: I gained experience in doing a "real" job with coding. Furthermore, this assignment highlighted the efficiency in using a regular expression, for example, the parsing vs normal string finding was very clear. I also got a great crash course in using regex commands and seeing firsthand how useful they can be when dealing with tedious situations. Also, formatting the output to come out in a certain way is definitely a good skill to have if I become a front-end programmer.

This was also one of the first times I used NOLINT in an assignment. I usually just changed my code, but some of the lines were really annoying to alter, so I just used NOLINT.



It's apparent that my solution worked; the rpt files were created, and the executable is show to work.

Makefile Mon Apr 19 19:54:05 2021 1

```
1: all: ps6
2:
3: ps6: main.o Kronos.o
4:         g++ -o ps6 Kronos.o main.o -lboost_regex -lboost_date_time
5:
6: main.o: main.cpp Kronos.hpp
7:         g++ -c main.cpp -Wall -ansi -pedantic
8:
9: Kronos.o: Kronos.cpp Kronos.hpp
10:         g++ -c Kronos.cpp -Wall -ansi -pedantic
11:
12: clean:
13:         rm *.o ps6
```

---

main.cpp Mon Apr 19 19:54:31 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #include <boost/regex.hpp>
4: #include <iostream>
5: #include <string>
6: #include <vector>
7: #include <fstream>
8: #include "Kronos.hpp"
9:
10: int main(int argc, char* argv[]) {
11:
12:     int i;  // initialize values
13:     int read = 0;
14:     int started = 0;
15:     int finished = 0;
16:
17:     std::ifstream fin;
18:     fin.open(argv[1]);
19:     // open file
20:
21:     std::string output_name(argv[1]);  // first arg
22:     output_name += ".rpt";  // create .rpt file
23:
```

```
24:     std::ofstream fout;
25:     fout.open(output_name.c_str());
26:     if (fout.fail()) {
27:         // ensure the opening of the file, throw error if not
28:         std::cerr << "Fail to output file: " << output_name << std::endl;
29:         return -1;
30:     }
31:
32:     std::string line;
33:     std::vector<InTouch> boots;
34:     while (std::getline(fin, line)) {
35:         read++;
36:         if (regex_match(line, startRegex)) {
37:             started++;
38:             InTouch its(line, argv[1], read);
39:             boots.push_back(its);
40:         } else if (regex_match(line, completedRegex)) {
41:             finished++;
42:             boots.back().completed(line, read);
43:         }
44:     }
45:
46:     for (i = 0; i < boots.size() - 1; ++i) { fout << boots[i] << std::endl; }
47:     fout << boots[boots.size() - 1];
48:     fin.close();
49:     fout.close();
50:     // close the files
51:
52:     return 0;
53: }
54:
```

Kronos.hpp Mon Apr 19 19:29:08 2021 1

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #ifndef _HOME_M_COMP4_PS6_KRONOS_HPP_
4: #define _HOME_M_COMP4_PS6_KRONOS_HPP_
5:
6: #include <stdio.h>
```

```
7: #include <string>
8: #include <boost/regex.hpp>
9:
10: const std::string start =  // NOLINT
11: "(\\d{4}\\-(\\d{2})\\-(\\d{2}) "
12: "(\\d{2}):(\\d{2}):(\\d{2})): "
13: "\\(log.c.166\\) server started\\s*";
14: const std::string completed =  // NOLINT
15: "(\\d{4}\\-(\\d{2})\\-(\\d{2}) "
16: "(\\d{2}):(\\d{2}):(\\d{2}))\\.(\\d{3})"
17: ".*oejs\\.AbstractConnector:Started\\s+"
18: "SelectChannelConnector@0.0.0.0:9080\\s*";
19: const boost::regex startRegex(start);
20: // get the start
21: const boost::regex completedRegex(completed);
22: // get the completed ones
23:
24: class InTouch {
25: // made a class tp make things easier on myself
26:  public:
27:     InTouch(std::string start_line, std::string file_name, int line);
28:     void completed(std::string completed, int line);
29:     int elapsedTime();
30:     // display elapsed time
31:     friend std::ostream& operator<<(std::ostream& out, const InTouch& right);
32:     // overload << operator
33:  private:
34:   // declare variables
35:     int _start;  // start
36:     int end;  // end
37:     int _boot_time;  // value for for elapsed time
38:     std::string _log_file_name;  // strings to pass
39:     std::string _start_time;
40:     std::string _end_time;
41:     bool _completed;
42: };
43:
44: #endif  // _HOME_M_COMP4_PS6_KRONOS_HPP_
45:
```

```
1: // Copyright [2021] <Meet Kothari>  [legal/copyright]
2:
3: #include <iostream>
4: #include <string>
5: #include <boost/date_time/posix_time/posix_time.hpp>
6: #include "Kronos.hpp"
7:
8: InTouch::InTouch(std::string start_line, std::string file_name, int line){
9:     _log_file_name = file_name;
10:     // update values
11:     _start = line;
12:     end = 0;
13:     // initialize
14:     _end_time = "";
15:     _boot_time = 0;
16:     _completed = false;
17:     boost::smatch sm;
18:     boost::regex_match(start_line, sm, startRegex);
19:     _start_time = sm[1];
20: }
21: void InTouch::completed(std::string completed, int line) {
22:     _completed = true;  // update values
23:     end = line;
24:     boost::smatch sm;
25:     boost::regex_match(completed, sm, completedRegex);
26:     // regex used to eval expression
27:     _end_time = sm[1];
28:     _boot_time = elapsedTime();
29: }
30: int InTouch::elapsedTime() {
31:     // display the elapsed time
32:     boost::posix_time::ptime start_time;
33:     boost::posix_time::ptime end_time;
34:     (boost::posix_time::time_from_string(_start_time));
35:     (boost::posix_time::time_from_string(_end_time));
36:     boost::posix_time::time_duration elapsedTime = end_time - start_time;
37:     return elapsedTime.total_milliseconds();
38: }
```

```
39: std::ostream& operator<<(std::ostream& out, const InTouch& right) {
40:     // user interface organization
41:     out << right._start << "(" << right._log_file_name << "): "
42:     << right._start_time << " Boot Start" << std::endl;
43:     if (right._completed) {
44:         out << right.end << "(" << right._log_file_name << "): "
45:         << right._end_time << " Boot Completed" << std::endl
46:         << "\tBoot Time: " << right._boot_time << "ms " << std::endl;
47:     } else {
48:         out << "**** Incomplete boot **** " << std::endl;
49:     }
50:     return out;
51: }
```