

Anson Cheang

COMP IV Sec 202: Project Portfolio

Spring 2022

**Contents:**

|  |        |
|--|--------|
| <b>PS0</b> Hello World with SFML                             | pg. 2  |
| <b>PS1</b> Linear Feedback Shift Register and Image Encoding | pg. 5  |
| <b>PS2</b> N-Body Simulation                                 | pg. 12 |
| <b>PS3</b> Recursive Graphics                                | pg. 23 |
| <b>PS4</b> CircularBuffer and StringSound                    | pg. 29 |
| <b>PS5</b> DNA Alignment                                     | pg. 40 |
| <b>PS6</b> Random Writer                                     | pg. 46 |
| <b>PS7</b> Kronos Time Parsing                               | pg. 54 |

Time to complete: 20 hours

## PS0: Hello World with SFML

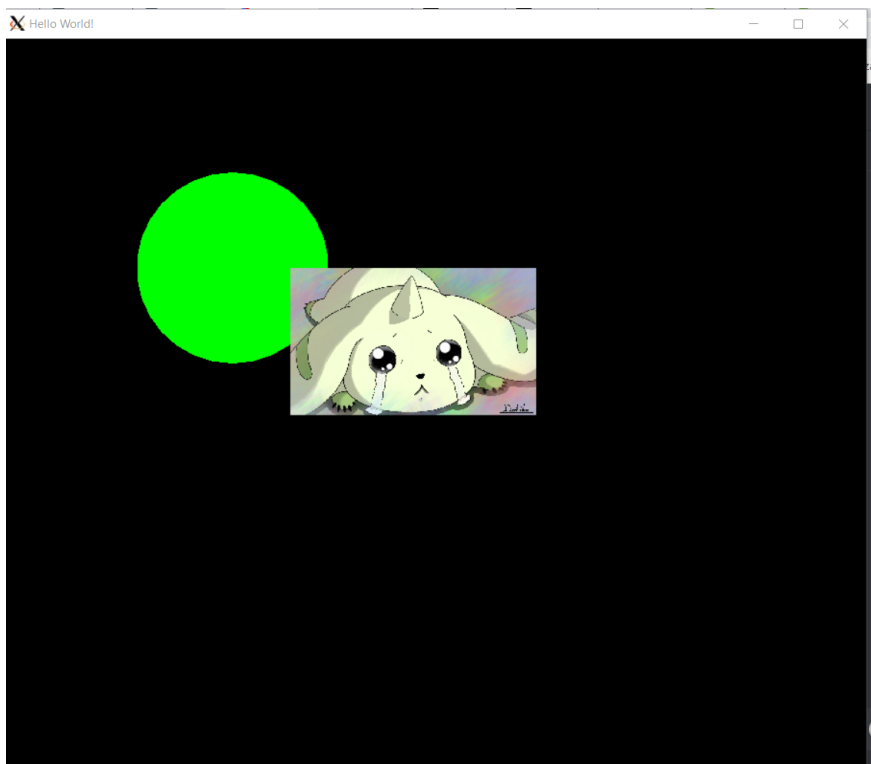
**Overview:** Essentially the main point of this assignment was to get us use to using sfml software and code. This assignment also have us get use to using Ubuntu, or VSCode, and XLaunch to make sure everyone has it and knows how to use it. Also made sure to download everything in advance. So for the output all we needed to do was produce a green circle, something that moves on keyboard command, a image sprite, and something else, I made mine rotate.

**Learned:** Well I learned how to use Ubuntu, and how the SFML window works. On the programming side I learned how to use some the of SFML codes, like how to display and draw out my sprite, and how to move it and set origins.

**Algorithm/Data Structures/OO Design:** The main algorithm is clear, draw, display in that order since that is needed for it to work.

**Difficulties:** The most difficult part of the assignment was the setup of the whole thing, with Ubuntu set up, and downloads, vscode, each download section etc. I got through this difficulty though not by myself, I had James Daly helping me with the downloading and set up. It took a bit to figue out how everything works. I learned how to set it up even though it took a while.

**Output:**



**Code:**

```
1: /**
2:  * main.cpp - testing out sfml, making them move and using the window
3:  *
4:  * Date 1/19/22 - 1/24/22
5:  *
6:  * Created by: Anson Cheang
7:  *
8:  */
9:
10: #include <SFML/Graphics.hpp>
11:
12: int main()
13: {
14:     float Gx = 10.f, Gy = 10.f;
15:     sf::RenderWindow window(sf::VideoMode(900, 800), "Hello World!");
16:
17:     window.setVerticalSyncEnabled(true);
18:     window.setFramerateLimit(15);
19:
20:     sf::CircleShape shape(100.f);
21:     shape.setFillColor(sf::Color::Green);
22:
23:     sf::Texture texture;
24:     if(!texture.loadFromFile("sprite.png"))
25:     {
26:         return EXIT_FAILURE;
27:     }
28:     sf::Sprite sprite(texture);
29:
30:     sprite.setScale(sf::Vector2f(.3, .3));
31:
32:     while (window.isOpen())
33:     {
34:         sf::Event event;
35:         while (window.pollEvent(event))
36:         {
37:             if (event.type == sf::Event::Closed)
38:                 window.close();
39:         }
40:
41:         shape.move(Gx, Gy);
42:         if(shape.getPosition().x >= 600.f)
43:         {
44:             Gx = -10.f;
45:             Gy = -10.f;
46:         }
47:         else if (shape.getPosition().x <= 0.f)
48:         {
49:             Gx = 10.f;
50:             Gy = 10.f;
51:         }
52:
53:         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left) && spr
te.getPosition().x > 0.f)
54:         {
55:             sprite.move(-10.f, 0.f);
56:         }
57:         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right) && spr
ite.getPosition().x < 600.f)
58:         {
59:             sprite.move(10.f, 0.f);
60:         }
61:         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down) && spr
ite.getPosition().y < 600.f)
62:         {
```

```
63:             sprite.move(0.f, 10.f);
64:         }
65:         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up) && sprite
.getPosition().y > 0.f)
66:         {
67:             sprite.move(0.f, -10.f);
68:         }
69:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))
70:         {
71:             sprite.rotate(90);
72:         }
73:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::F))
74:         {
75:             sprite.setScale(sf::Vector2f(.3, -.3));
76:         }
77:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::R))
78:         {
79:             sprite.setScale(sf::Vector2f(.3, .3));
80:         }
81:
82:         window.clear();
83:         window.draw(shape);
84:         window.draw(sprite);
85:         window.display();
86:     }
87:
88:     return 0;
89: }
```

## PS1: Linear Feedback Shift Register and Image Encoding

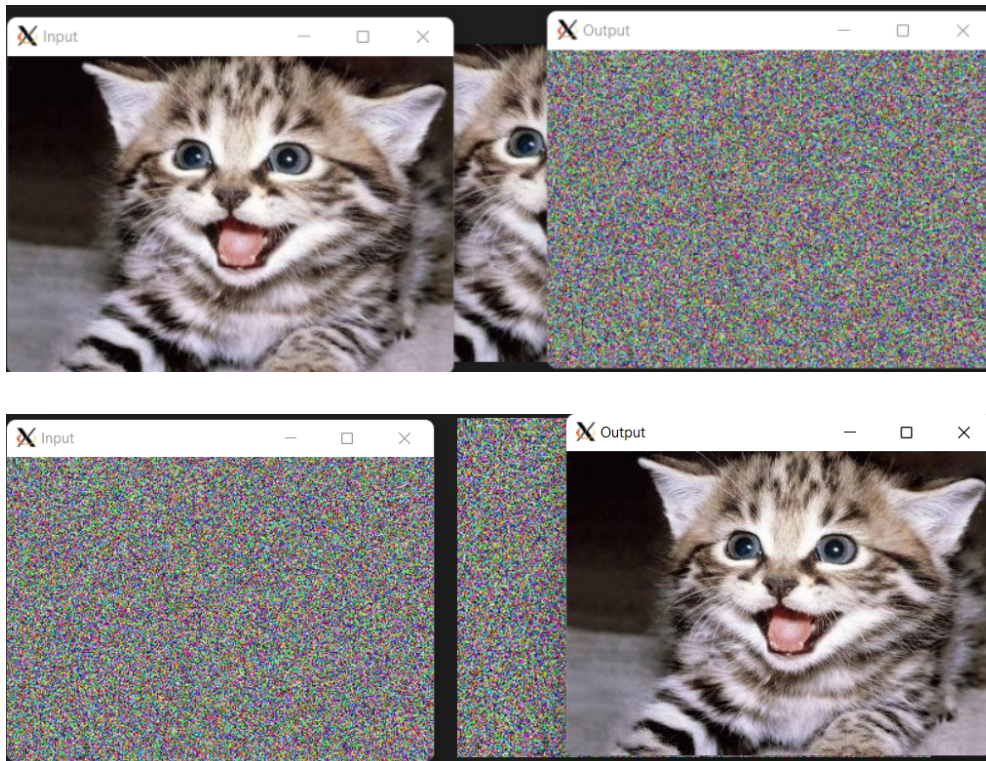
**Overview:** The Assignment is to encode and decode a certain image, in my case the cat.jpg within the attached files section of the ps1b assignment. To do this we pass in an input-file.png, output-file.png, and a string of bits to encode and decode that image. With the string of bit, we pass that in to make a FibLFSR class object variable, and convert the image into static when encoding and while decoding turn the static back into the original image. To do this we essentially use generate to generate a number and change each individual pixel on the red, blue, yellow spectrum.

**Learned:** I learned how to program in and change the color of an image sprite to be used later. I also learned how LFSR works, and should be able to use it in the future.

**Algorithm/Data Structures/OO Design:** A key algorithm is the transform function, since it allows me to encode and decode the image that was inputted, and the LSFR algorithm, because otherwise this entire thing would not have worked in the first place

**Difficulties:** Well the code works only when encoding the image, not decoding it, I overcame it and fixed the issue so it works now. Overall nothing much to fix therefore nothing much to learn, did help me get use to debugging though.

**Output:**



**Code:**

```
1: CC = g++
2: CFLAGS = -Wall -Werror -pedantic --std=c++14
3: LIBS = -lboost_unit_test_framework
4: DEPS = FibLFSR.h
5: SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
6:
7: %.o: %.cpp $(DEPS)
8:     $(CC) $(CFLAGS) -c $<
9:
10: all: PhotoMagic
11:
12: PhotoMagic: photoMagic.o FibLFSR.o
13:     $(CC) $(CFLAGS) -o PhotoMagic $^ $(LIBS) $(SFMLFLAGS)
14:
15: clean:
16:     rm *.o PhotoMagic
```

```
1: /**
2:  * photoMagic.cpp - Essentially to encode and decode an image using the Fi
bLFSR
3:  * that was programmed back in ps1a as an assignment
4:  *
5:  * Date 2/1/22 - 2/7/22
6:  *
7:  * Created by: Anson Cheang
8:  *
9:  */
10:
11: #include <SFML/System.hpp>
12: #include <SFML/Window.hpp>
13: #include <SFML/Graphics.hpp>
14: #include "FibLFSR.h"
15:
16: // transforms image using FibLFSR
17: void transform( sf::Image&, FibLFSR*);
18:
19: // display an encrypted copy of the picture, using the LFSR
20: // to do the encryption
21: int main(int argc, char* argv[])
22: {
23:     sf::Image image1;
24:     FibLFSR encryptionCode(argv[3]);
25:     if (!image1.loadFromFile(argv[1]))
26:     {
27:         return -1;
28:     }
29:
30:     sf::Vector2u size = image1.getSize();
31:     sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Input");
32:
33:     sf::Texture texture;
34:     texture.loadFromImage(image1);
35:
36:     sf::Sprite sprite;
37:     sprite.setTexture(texture);
38:
39:     transform(image1, &encryptionCode);
40:
41:     sf::Vector2u size2 = image1.getSize();
42:     sf::RenderWindow window2(sf::VideoMode(size2.x, size2.y), "Output
");
43:
44:     sf::Texture texture2;
45:     texture2.loadFromImage(image1);
46:
47:     sf::Sprite sprite2;
48:     sprite2.setTexture(texture2);
49:
50:     while (window1.isOpen() && window2.isOpen())
51:     {
52:         sf::Event event;
53:         while (window1.pollEvent(event)) {
54:             if (event.type == sf::Event::Closed)
55:             {
56:                 window1.close();
57:             }
58:         }
59:         while (window2.pollEvent(event))
60:         {
61:             if (event.type == sf::Event::Closed)
62:             {
63:                 window2.close();
```

```
64:         }
65:     }
66:     window1.clear();
67:     window1.draw(sprite);
68:     window1.display();
69:     window2.clear();
70:     window2.draw(sprite2);
71:     window2.display();
72: }
73:
74: if (!image1.saveToFile(argv[2]))
75: {
76:     return -1;
77: }
78:
79: return 0;
80: }
81:
82: void transform( sf::Image& image, FibLFSR* encryptionCode)
83: {
84:     sf::Color p;
85:     sf::Vector2u size = image.getSize();
86:
87:     // create photographic negative image of upper-left 200 px square
88:     for (int x = 0; x < static_cast<int>(size.x); x++) {
89:         for (int y = 0; y < static_cast<int>(size.y); y++) {
90:             p = image.getPixel(x, y);
91:             p.r = p.r ^ encryptionCode->generate(8);
92:             p.g = p.g ^ encryptionCode->generate(8);
93:             p.b = p.b ^ encryptionCode->generate(8);
94:             image.setPixel(x, y, p);
95:         }
96:     }
97: }
```



```
1: #ifndef FibLFSR_H_
2: #define FibLFSR_H_
3: #include <string>
4: #include <vector>
5: #include <iostream>
6:
7: using namespace std;
8:
9: class FibLFSR {
10: public:
11:     FibLFSR(string seed);    // constructor to create LFSR with
12:                             // the given initial seed
13:     int step();              // simulate one step and return the
14:                             // new bit as 0 or 1
15:     int generate(int k);     // simulate k steps and return
16:                             // k-bit integer
17:     friend ostream& operator<<(ostream& out, FibLFSR CurrentBits);
18: private:
19:     vector<int> list;
20: };
21:
22: #endif
```

```
1: /**
2:  * FibLFSR.cpp - To run the FubLFSR class, in which they constructor, step
3:  * and generator(), essentially using Linear feedback shift register
4:  *
5:  * Date 1/24/22 - 1/31/22
6:  *
7:  * Created by: Anson Cheang
8:  *
9:  */
10:
11: #include "FibLFSR.h"
12: #include <vector>
13: #include <string>
14: #include <cmath>
15:
16: using namespace std;
17:
18: FibLFSR::FibLFSR(string seed)
19: {
20:     string Pbit;
21:     for(int i = 0; i < static_cast <int> (seed.length()); i++)
22:     {
23:         Pbit = seed[i];
24:         list.push_back(stoi(Pbit, 0, 10));
25:     }
26: }
27:
28:
29: int FibLFSR::step()
30: {
31:     int size = static_cast <int> (list.size());
32:     int RBit = list[0] ^ list[2] ^ list[3] ^ list[5];
33:     for(int i = 0; i < size - 1; i++)
34:     {
35:         list[i] = list[i+1];
36:     }
37:
38:     list[size - 1] = RBit;
39:
40:     return RBit;
41: }
42:
43: int FibLFSR::generate(int k)
44: {
45:     if(k > 32 || k < 0)
46:     {
47:         throw out_of_range("The value inputted isn't within the range of
0 - 32");
48:     }
49:     int RBit;
50:     int total = 0;
51:     for(int i = 0; i < k; i++)
52:     {
53:         RBit = step();
54:         total = total + pow(2, k-i-1) * RBit;
55:     }
56:
57:     return total;
58: }
59:
60: ostream& operator<<(ostream& out, FibLFSR CurrentBits)
61: {
62:     vector<int>::iterator it;
63:     for(it = CurrentBits.list.begin(); it != CurrentBits.list.end(); it++)
```

```
)  
64:     {  
65:         out << *it;  
66:     }  
67:     return out;  
68: }
```

## PS2: N-Body Simulation

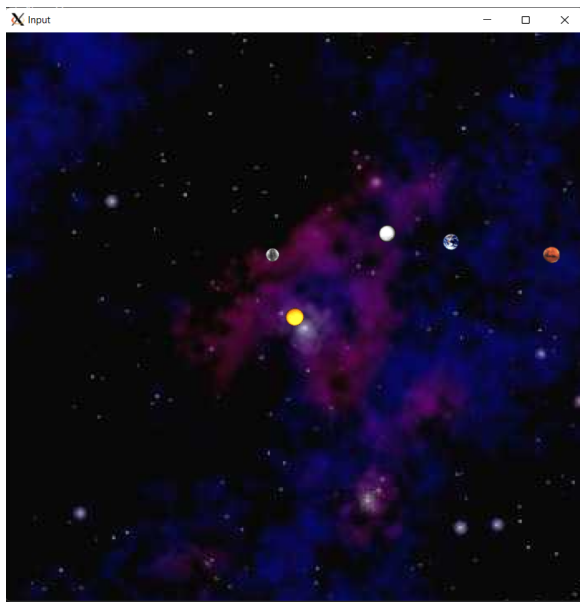
**Overview:** This assignment requires us to make a simulation of bodies moving similar to how they would be in a solar system. We first needed to implement several equations needed to make sure that the force that each celestial body emit on each other is calculated, along with its pathing to produce the correct placement of each body after a certain amount of time. Then we need to use that whole function to make the body move along the window, matching the equation given, making it look accurate to how planets are supposed to move.

**Learned:** So I learned physics in how each planets move within this simulations. I also learned overloading draw functions and massive movement of objects in SFML.

**Algorithm/Data Structures/OO Design:** Starting off a OO Design would be a vector of unique pointers of CelestialBody storage, as a way to store each individual CelestialBody object just so I can change them as I need be through each pointer. And a key algorithm would be the step functions since it does a lot in order to keep the program working with the planets it uses

**Difficulties:** The planets keep disappearing when outputting, which was fixed since I was apparently doing self-reference cout and it was having some issues, it was fixed however, primarily thanks to professor Daly, apparently the friend operator was creating copies of the unique pointer. So I learned that you have to reference the unique pointer, since if not it makes a copy of it breaking the code.

**Output:**



**Code:**

```
1: CC = g++
2: CFLAGS = -Wall -Werror -pedantic --std=c++14
3: LIBS = -lboost_unit_test_framework
4: DEPS = CelestialBody.h Universe.h
5: SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
6:
7: %.o: %.cpp $(DEPS)
8:     $(CC) $(CFLAGS) -c $<
9:
10: all: NBody
11:
12: NBody: main.o CelestialBody.o Universe.o
13:     $(CC) $(CFLAGS) -o NBody $^ $(LIBS) $(SFMLFLAGS)
14:
15: clean:
16:     rm *.o NBody
```

```
1: /**
2:  * main.cpp - as a base to run the program
3:  *
4:  * Date 2/14/22 - 2/22/22
5:  *
6:  * Created by: Anson Cheang
7:  *
8:  */
9:
10: /*#include <SFML/System.hpp>
11: #include <SFML/Window.hpp>
12: #include <SFML/Graphics.hpp>*/
13: #include "CelestialBody.h"
14: #include "Universe.h"
15: #include <iostream>
16: #include <cstdlib>
17:
18: using namespace std;
19:
20: int main(int argc, char* argv[])
21: {
22:     double time = atoi(argv[1]);
23:     double seconds = 0;
24:     sf::RenderWindow window(sf::VideoMode(700, 700), "Input");
25:
26:     window.setVerticalSyncEnabled(true);
27:     window.setFramerateLimit(15);
28:
29:     sf::Image image;
30:     if(!image.loadFromFile("starfield.jpg"))
31:     {
32:         return -1;
33:     }
34:     sf::Texture texture;
35:     texture.loadFromImage(image);
36:     sf::Sprite sprite;
37:     sprite.setTexture(texture);
38:     sf::Vector2u size = image.getSize();
39:     sprite.setScale((1+700/size.x), (1+700/size.y));
40:
41:     int amount;
42:
43:     cin >> amount;
44:
45:     Universe space(amount);
46:
47:     while (window.isOpen())
48:     {
49:         sf::Event event;
50:         while (window.pollEvent(event))
51:         {
52:             if (event.type == sf::Event::Closed)
53:             {
54:                 window.close();
55:             }
56:         }
57:
58:         window.clear();
59:         window.draw(sprite);
60:         if(seconds <= time)
61:         {
62:             space.step(atoi(argv[2]));
63:             seconds += atoi(argv[2]);
64:         }
65:         window.draw(space);
```

```
66:         window.display();
67:     }
68:     cout << space;
69:     return 0;
70: }
```

```
1: #ifndef Universe_H_
2: #define Universe_H_
3:
4: #include <SFML/System.hpp>
5: #include <SFML/Window.hpp>
6: #include <SFML/Graphics.hpp>
7: #include <vector>
8: #include <iostream>
9: #include "CelestialBody.h"
10:
11: using namespace std;
12:
13: class Universe : public sf::Drawable
14: {
15: public:
16:     Universe(int size);
17:     void step(double seconds);
18:     friend ostream& operator<<(ostream& out, const Universe& Galaxy);
19: private:
20:
21:     void draw(sf::RenderTarget& target, sf::RenderStates states) const;
22:     int galaxySize;
23:     double maxR;
24:     vector<unique_ptr<CelestialBody> > galaxy;
25: };
26:
27: ostream& operator<<(ostream& out, const Universe& Galaxy);
28:
29: #endif
```



```
1: /**
2: * Universe.cpp - as an implementation to store every CelestialBody object
3: * in order to have them created, and draw them out, essentially storage
4: * and also to do the physics to make each individual particle move in rot
ation
5: *
6: * Date 2/14/22 - 2/22/22
7: *
8: * Created by: Anson Cheang
9: *
10: */
11:
12: #include "Universe.h"
13: #include <cmath>
14:
15: Universe::Universe(int size)
16: {
17:     //double posX, posY, Xvel, Yvel, Imass;
18:     double radius, scale;
19:     galaxySize = size;
20:     //string filename;
21:     cin >> radius;
22:     maxR = radius;
23:     scale = 350/radius;
24:     //double scaledXPos, scaledYPos;
25:     for(int i = 0; i < size; i++)
26:     {
27:         galaxy.push_back(make_unique<CelestialBody>(scale));
28:         cin >> *(galaxy[i]);
29:         galaxy[i]->createImage();
30:         //cin >> posX >> posY >> Xvel >> Yvel >> Imass >> filename;
31:         //scaledXPos = posX*scale + 350;
32:         //scaledYPos = posY*scale + 350;
33:         //galaxy.push_back(make_unique<CelestialBody>(scaledXPos, scaledY
Pos, Xvel, Yvel, Imass, filename));
34:     }
35: }
36:
37: void Universe::draw(sf::RenderTarget& target, sf::RenderStates states) co
nst
38: {
39:     for(int i = 0; i < galaxySize; i++)
40:     {
41:         target.draw(*(galaxy[i]), states);
42:     }
43: }
44:
45: void Universe::step(double second)
46: {
47:     sf::Vector2f netForce;
48:     sf::Vector2f Accelleration;
49:     sf::Vector2f velocity;
50:     sf::Vector2f p;
51:     double CX, CY;
52:     double radius;
53:     double grav = 6.67e-11;
54:     double force;
55:     for(int i = 0; i < galaxySize; i++)
56:     {
57:         netForce.x = 0;
58:         netForce.y = 0;
59:         for(int j = 0; j < galaxySize; j++)
60:         {
61:             if(i != j)
62:             {
```

```
63:             CX = galaxy[i]->getXPos()-galaxy[j]->getXPos();
64:             CY = galaxy[i]->getYPos()-galaxy[j]->getYPos();
65:             radius = sqrt(pow(CX, 2) + pow(CY, 2));
66:             force = (grav * galaxy[i]->getMass() * galaxy[j]->getMass
()) / pow(radius, 2);
67:             netForce.x += force * (CX/radius);
68:             netForce.y += force * (CY/radius);
69:         }
70:     }
71:     Accelleration.x = netForce.x/galaxy[i]->getMass();
72:     Accelleration.y = netForce.y/galaxy[i]->getMass();
73:
74:     velocity.x = galaxy[i]->getXVel() + second * Accelleration.x;
75:     velocity.y = galaxy[i]->getYVel() + second * Accelleration.y;
76:     galaxy[i]->setVel(velocity);
77:
78:     p.x = galaxy[i]->getXPos() + -(second * velocity.x);
79:     p.y = galaxy[i]->getYPos() + -(second * velocity.y);
80:     galaxy[i]->setPos(p);
81:     galaxy[i]->setImagePos();
82: }
83: }
84:
85: ostream& operator<<(ostream& out, const Universe& space)
86: {
87:     out << space.galaxySize << endl;
88:     out << space.maxR << endl;
89:     for(int i = 0; i < space.galaxySize; i++)
90:     {
91:         out << *(space.galaxy[i]);
92:     }
93:     return out;
94: }
```

```
1: #ifndef CelestialBody_H_
2: #define CelestialBody_H_
3:
4: #include <SFML/System.hpp>
5: #include <SFML/Window.hpp>
6: #include <SFML/Graphics.hpp>
7: #include <string>
8: #include <cstdlib>
9: #include <iostream>
10:
11: using namespace std;
12:
13: class CelestialBody : public sf::Drawable
14: {
15: public:
16:
17:     CelestialBody(double val);
18:     void createImage();
19:
20:     CelestialBody(double posX, double posY, double Xvel, double Yvel, double
Imass, string _filename);
21:     friend istream& operator>>(istream& instream, CelestialBody& planet);
22:     friend ostream& operator<<(ostream& out, CelestialBody planet);
23:
24:     void setPos(sf::Vector2f Pos);
25:     void setVel(sf::Vector2f Vel);
26:     void setImagePos();
27:     double getXPos();
28:     double getYPos();
29:     double getMass();
30:     double getXVel();
31:     double getYVel();
32:
33: private:
34:
35:     void draw(sf::RenderTarget& target, sf::RenderStates states) const;
36:     double XPosition;
37:     double YPosition;
38:     double XVelocity;
39:     double YVelocity;
40:     double Mass;
41:     double scale;
42:     string filename;
43:     sf::Image image;
44:     sf::Texture texture;
45:     sf::Sprite sprite;
46: };
47:
48:
49: istream& operator>>(istream& instream, CelestialBody& planet);
50: ostream& operator<<(ostream& out, CelestialBody planet);
51:
52: #endif
```

```
1: /**
2:  * CelestialBody.cpp - an implementation to create each celestial body
3:  * 1 at a time, and also place them into the correct location
4:  * for drawing. plus draw each one individually, and overode >> operator
5:  *
6:  * Date 2/14/22 - 2/22/22
7:  *
8:  * Created by: Anson Cheang
9:  *
10: */
11:
12: #include "CelestialBody.h"
13: #include <SFML/System.hpp>
14: #include <SFML/Window.hpp>
15: #include <SFML/Graphics.hpp>
16: #include <string>
17: #include <cstdlib>
18: #include <iostream>
19:
20: using namespace std;
21:
22: CelestialBody::CelestialBody(double val)
23: {
24:     scale = val;
25:     XPosition = 0;
26:     YPosition = 0;
27:     XVelocity = 0;
28:     YVelocity = 0;
29:     Mass = 0;
30:     filename = "";
31: }
32:
33:
34: void CelestialBody::createImage()
35: {
36:     if(!image.loadFromFile(filename))
37:     {
38:         exit(-1);
39:     }
40:
41:     texture.loadFromImage(image);
42:
43:     sprite.setTexture(texture);
44:     sf::Vector2u size = image.getSize();
45:     sprite.setOrigin(static_cast<int>(size.x)/2, static_cast<int>(size.y)
/2);
46:     sprite.setPosition(sf::Vector2f(XPosition*scale + 350, YPosition*scale
e + 350));
47: }
48:
49: CelestialBody::CelestialBody(double posX, double posY, double Xvel, doubl
e Yvel, double Imass, string _filename)
50: {
51:     XPosition = posX;
52:     YPosition = posY;
53:     XVelocity = Xvel;
54:     YVelocity = Yvel;
55:     Mass = Imass;
56:     filename = _filename;
57:
58:     if(!image.loadFromFile(filename))
59:     {
60:         exit(-1);
61:     }
62:
```

```
63:     texture.loadFromImage(image);
64:
65:     sprite.setTexture(texture);
66:     sf::Vector2u size = image.getSize();
67:     sprite.setOrigin(static_cast<int>(size.x)/2, static_cast<int>(size.y)
/2);
68:     sprite.setPosition(sf::Vector2f(posX, posY));
69: }
70:
71: void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates state
s) const
72: {
73:     target.draw(sprite, states);
74: }
75:
76: istream& operator>>(istream& instream, CelestialBody& planet)
77: {
78:     instream >> planet.XPosition >> planet.YPosition >> planet.XVelocity
>> planet.YVelocity >> planet.Mass >> planet.filename;
79:     return instream;
80: }
81:
82: void CelestialBody::setPos(sf::Vector2f Pos)
83: {
84:     XPosition = Pos.x;
85:     YPosition = Pos.y;
86: }
87:
88: void CelestialBody::setVel(sf::Vector2f Vel)
89: {
90:     XVelocity = Vel.x;
91:     YVelocity = Vel.y;
92: }
93:
94: double CelestialBody::getXPos()
95: {
96:     return XPosition;
97: }
98:
99: double CelestialBody::getYPos()
100: {
101:     return YPosition;
102: }
103:
104: double CelestialBody::getMass()
105: {
106:     return Mass;
107: }
108:
109: void CelestialBody::setImagePos()
110: {
111:     //double CX = (Pos.x - XPosition) * scale;
112:     //double CY = (Pos.y - YPosition) * scale;
113:     sprite.setPosition(sf::Vector2f(XPosition*scale + 350, YPosition*scale
e + 350));
114:     //cout << XPosition*scale + 350 << ", " << YPosition*scale + 350 << e
ndl;
115:     //XPosition = Pos.x;
116:     //YPosition = Pos.y;
117: }
118:
119: double CelestialBody::getXVel()
120: {
121:     return XVelocity;
122: }
```

```
123:
124: double CelestialBody::getYVel()
125: {
126:     return YVelocity;
127: }
128:
129:
130: ostream& operator<<(ostream& out, CelestialBody planet)
131: {
132:     out << planet.XPosition << " " << planet.YPosition << " " << planet
.XVelocity << " "
133:         << planet.YVelocity << " " << planet.Mass << " " << planet.file
name << endl;
134:     return out;
135: }
```

### PS3: Recursive Graphics

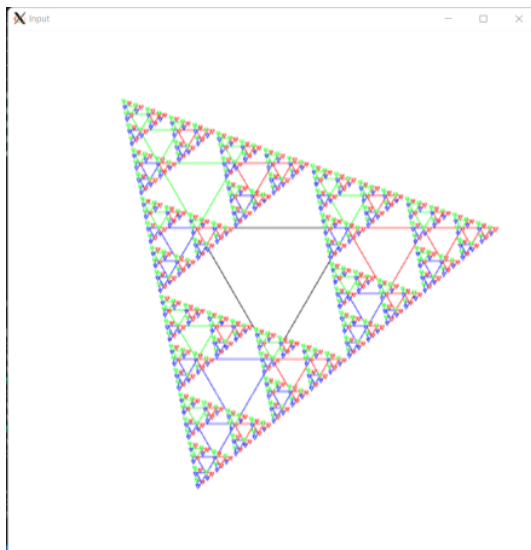
**Overview:** The assignment requires us to use recursive to draw out a triangle surrounded by smaller triangle. it is essentially a sierpinski triangle but reversed, so rather than the smaller triangle on the inside the smaller triangle is on the outside instead making a weird triangle design. Overall I used curvex shape to design my triangle and positioned each triangle to its proper position in order for them to be drawn. I also changed the color, so top left is green, top right is red, and bottom is blue.

**Learned:** Recursive, and that trees can have 3 children instead of 2, which I always been shown it.

**Algorithm/Data Structures/OO Design:** the recursive function is a key algorithms, since it was needed to design the whole triangle shape and design, and also made it so it only repeat as needed without making it so that you need some repeated code over and over again of course the recursive is called 3 times per function if it pass the if statement check since each point of the triangle requires a triangle, and each of those triangle requires 3 triangle as well so yeah

**Difficulties:** The window would occasionally show a blank screen, with nothing being drawn on them. It wasn't an issue with the code, since when testing anytime it was a under 10 it works, and when it was at least 10, it would take too long. Apparently my code takes some time to run the code, and as such the higher the number the longer it takes to completely draw everything. So there wasn't much to fix just processing speed. Although did teach me to sometimes wait to see if it just taking a while.

**Output:**



**Code:**

```
1: CC = g++
2: CFLAGS = -Wall -Werror -pedantic --std=c++14
3: LIBS = -lboost_unit_test_framework
4: DEPS = Triangle.h
5: SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
6:
7: %.o: %.cpp $(DEPS)
8:     $(CC) $(CFLAGS) -c $<
9:
10: all: TFractal
11:
12: TFractal: TFractal.o Triangle.o
13:     $(CC) $(CFLAGS) -o TFractal $^ $(LIBS) $(SFMLFLAGS)
14:     cpplint --filter=--runtime/references *.cpp *.h
15:
16: clean:
17:     rm *.o TFractal
```



```
1: // Copyright 2022 Anson Cheang
2: /**
3:  * TFractal.cpp - essentially the main function,
4:  * which calls upon the recursive function to create
5:  * each new triangle and draw them out in different color
6:  *
7:  * Date 2/22/22 - 2/28/22
8:  *
9:  * Created by: Anson Cheang
10:  *
11:  */
12: #include "Triangle.h"
13: #include <iostream>
14: #include <cstdlib>
15: #include <cmath>
16: #include <SFML/System.hpp>
17: #include <SFML/Window.hpp>
18: #include <SFML/Graphics.hpp>
19:
20: // using namespace std;
21:
22: void fTree(sf::RenderWindow& window, Triangle ET, double size, int depth)
;
23:
24: int main(int argc, char* argv[]) {
25:     /*double windowSize = atoi(argv[1]);
26:     double currentSize = atoi(argv[1]);
27:     for(int i = 1; i < atoi(argv[2]); i++)
28:     {
29:         currentSize = (sqrt(3)/4) * pow(currentSize, 2);
30:         currentSize = currentSize/4;
31:         currentSize = currentSize * 4/sqrt(3);
32:         currentSize = sqrt(currentSize);
33:         windowSize += currentSize;
34:     }*/
35:
36:     sf::RenderWindow window(sf::VideoMode(700, 700), "Input");
37:     double height = sqrt(3)/2*atoi(argv[1]);
38:     sf::Vector2f position;
39:     position.x = (700/2) - atoi(argv[1])/2;
40:     position.y = (700/2) - (height/2);
41:     Triangle triangle(atoi(argv[1]), position, 'n');
42:
43:     while (window.isOpen()) {
44:         sf::Event event;
45:         while (window.pollEvent(event)) {
46:             if (event.type == sf::Event::Closed) {
47:                 window.close();
48:             }
49:         }
50:
51:         window.clear(sf::Color::White);
52:         fTree(window, triangle, atoi(argv[1]), atoi(argv[2]));
53:         // window.draw(triangle);
54:         window.display();
55:     }
56:
57:     return 0;
58: }
59:
60: void fTree(sf::RenderWindow& window, Triangle ET, double size, int depth)
{
61:     window.draw(ET);
62:     if (depth > 0) {
63:         sf::Vector2f position = ET.getP1();
```

```
64:         size = size/2;
65:         position.y = position.y - size * sqrt(3) / 2;
66:         position.x = position.x - size/2;
67:         Triangle T1(size, position, 'g');
68:         fTree(window, T1, size, depth - 1);
69:         // position = ET.getP2();
70:         Triangle T2(size, ET.getP2(), 'r');
71:         fTree(window, T2, size, depth - 1);
72:         position = ET.getP3();
73:         position.x = position.x - size;
74:         Triangle T3(size, position, 'b');
75:         fTree(window, T3, size, depth - 1);
76:     }
77: }
```

```
1: // Copyright 2022 Anson Cheang
2: #ifndef _HOME_IIFORCE_BADNAME_COMP4_PS3_TRIANGLE_H_ // Triangle_H_
3: #define _HOME_IIFORCE_BADNAME_COMP4_PS3_TRIANGLE_H_ // Triangle_H_
4:
5: #include <string>
6: #include <cstdlib>
7: #include <iostream>
8: #include <SFML/System.hpp>
9: #include <SFML/Window.hpp>
10: #include <SFML/Graphics.hpp>
11:
12: class Triangle : public sf::Drawable{
13: public:
14: Triangle(double val, sf::Vector2f position, char color);
15: sf::Vector2f getP1();
16: sf::Vector2f getP2();
17: sf::Vector2f getP3();
18:
19: private:
20: void draw(sf::RenderTarget& target, sf::RenderStates states) const;
21: double size;
22: sf::Vector2f P1;
23: sf::Vector2f P2;
24: sf::Vector2f P3;
25: sf::ConvexShape shape;
26: };
27:
28:
29: #endif // _HOME_IIFORCE_BADNAME_COMP4_PS3_TRIANGLE_H_
```

```
1: // Copyright 2022 Anson Cheang
2: /**
3:  * Triangle.cpp - as an implementation to create a new triangle object to
store every point
4:  * and draw out the triangle at a moments notice
5:  *
6:  * Date 2/22/22 - 2/28/22
7:  *
8:  * Created by: Anson Cheang
9:  *
10: */
11: #include "Triangle.h"
12: #include <string>
13: #include <cstdlib>
14: #include <iostream>
15: #include <cmath>
16: #include <SFML/System.hpp>
17: #include <SFML/Window.hpp>
18: #include <SFML/Graphics.hpp>
19:
20: // using namespace std;
21:
22:
23: Triangle::Triangle(double val, sf::Vector2f position, char color) {
24:     size = val;
25:     sf::Vector2f point1 = position, point2 = position;
26:     point1.x = point1.x + val;
27:     point2.x = (position.x + point1.x)/2;
28:     point2.y = point2.y + sqrt(3)/2 * val;
29:     P1 = position;
30:     P2 = point1;
31:     P3 = point2;
32:     shape.setPointCount(3);
33:     shape.setPoint(0, position);
34:     shape.setPoint(1, point1);
35:     shape.setPoint(2, point2);
36:     shape.setOutlineThickness(1);
37:     if (color == 'g') {
38:         shape.setOutlineColor(sf::Color::Green);
39:     } else if (color == 'r') {
40:         shape.setOutlineColor(sf::Color::Red);
41:     } else if (color == 'b') {
42:         shape.setOutlineColor(sf::Color::Blue);
43:     } else {
44:         shape.setOutlineColor(sf::Color::Black);
45:     }
46: }
47:
48: sf::Vector2f Triangle::getP1() {
49:     return P1;
50: }
51:
52: sf::Vector2f Triangle::getP2() {
53:     return P2;
54: }
55:
56: sf::Vector2f Triangle::getP3() {
57:     return P3;
58: }
59:
60: void Triangle::draw(sf::RenderTarget& target, sf::RenderStates states) co
nst {
61:     target.draw(shape, states);
62: }
```

## PS4: CircularBuffer and StringSound

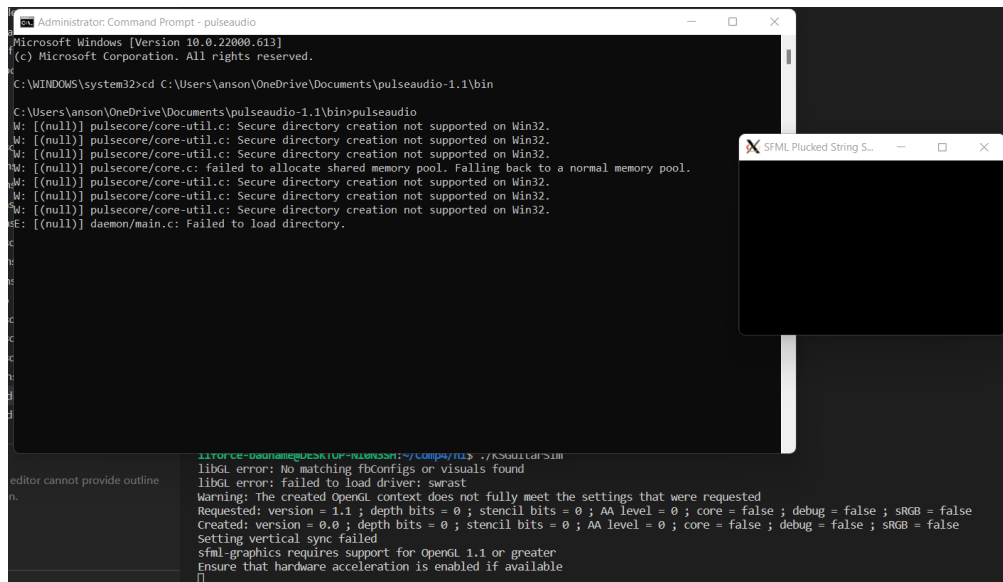
**Overview:** So the purpose of this assignment is to make it so that you can produce sounds similar to that of a guitar except from the computer. In order to do this we had to first make the circular buffer, in which will dictate the sound length, and change as the sound gets played, similar to when plucking the guitar. Then we had to set up the noise per keyboard input, there is only a set amount, and only those amount could be inputted otherwise quit the program, and those inputs have their own set of unique noises, on different pitched

**Learned:** I learned how to use pulse audio, and how to set up the sound system within my code.

**Algorithm/Data Structures/OO Design:** Well there are lambda expressions and exceptions within the code. The lambda is there mainly because it was a requirement for the project, plus I couldn't really think of a place where I should have placed it. There is also a vector of circular buffers. This is mainly to store each note, and as such I would always be able to have every note prepared beforehand

**Difficulties:** Audiopulse was difficult to set up. I did learn how to set it up thanks to James Daly's help. Another difficulty was segment fault, which was because I did not set up the constructor correctly. Then audio sounds the same, its the sound doesnt store buffer, and sound buffer doesnt store sample which was fixed, made the vector of vector of samples first, then use the for loop to make a vector of unique pointer of soundBuffer, and a third for to make a vector of sound from the soundBuffer. So I learned how audiosound it supposed to work, plus its stupid way of doing it

**Output:** It only plays sound so here is a screenshot of it working



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt - pulseaudio" and an SFML application window titled "SFML Plucked String S...". The Command Prompt shows the execution of the pulseaudio command and various error messages from the pulsecore and daemon processes. The SFML application window shows a black screen with a small red 'X' icon in the top left corner, indicating an error or crash.

```
Administrator: Command Prompt - pulseaudio
Microsoft Windows [Version 10.0.22000.612]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd C:\Users\anson\OneDrive\Documents\pulseaudio-1.1\bin

C:\Users\anson\OneDrive\Documents\pulseaudio-1.1\bin>pulseaudio
W: [null] pulsecore/core-util.c: Secure directory creation not supported on Win32.
W: [null] pulsecore/core-util.c: Secure directory creation not supported on Win32.
W: [null] pulsecore/core-util.c: Secure directory creation not supported on Win32.
W: [null] pulsecore/core.c: failed to allocate shared memory pool. Falling back to a normal memory pool.
W: [null] pulsecore/core-util.c: Secure directory creation not supported on Win32.
W: [null] pulsecore/core-util.c: Secure directory creation not supported on Win32.
W: [null] pulsecore/core-util.c: Secure directory creation not supported on Win32.
W: [null] daemon/main.c: Failed to load directory.

libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
Warning: The created OpenGL context does not fully meet the settings that were requested
Requested: version = 1.1 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Created: version = 0.0 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Setting vertical sync failed
sfml-graphics requires support for OpenGL 1.1 or greater
Ensure that hardware acceleration is enabled if available
```

**Code:**

```
1: CC = g++
2: CFLAGS = -Wall -Werror -pedantic --std=c++14
3: LIBS = -lboost_unit_test_framework
4: DEPS = CircularBuffer.h StringSound.h
5: SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
6:
7: %.o: %.cpp $(DEPS)
8:     $(CC) $(CFLAGS) -c $<
9:
10: all: KSGuitarSim
11:
12: KSGuitarSim: KSGuitarSim.o CircularBuffer.o StringSound.o
13:     $(CC) -g $(CFLAGS) -o KSGuitarSim $^ $(LIBS) $(SFMLFLAGS)
14:     cpplint --filter=--runtime/references *.cpp *.h
15:
16: clean:
17:     rm *.o KSGuitarSim
```

```
1:  /*
2:   Copyright 2015 Fred Martin,
3:   Y. Rykalova, 2020
4:   J. Daly 2022
5:
6:   Edited by Anson Cheang 2022
7:   essentially allows the user to play
8:   a unique sound from 37 different keys.
9:   the function automatically sets up the sounds
10: */
11:
12: #include "CircularBuffer.h"
13: #include "StringSound.h"
14:
15: #include <math.h>
16: #include <limits.h>
17:
18: #include <iostream>
19: #include <string>
20: #include <exception>
21: #include <stdexcept>
22: #include <vector>
23:
24: #include <SFML/Graphics.hpp>
25: #include <SFML/System.hpp>
26: #include <SFML/Audio.hpp>
27: #include <SFML/Window.hpp>
28:
29: #define CONCERT_A 220.0
30: #define SAMPLES_PER_SEC 44100
31:
32: // using namespace std;
33:
34: std::vector<sf::Int16> makeSamples(StringSound& gs) {
35:     std::vector<sf::Int16> samples;
36:
37:     gs.pluck();
38:     int duration = 8; // seconds
39:     int i;
40:     for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
41:         gs.tic();
42:         samples.push_back(gs.sample());
43:     }
44:
45:     return samples;
46: }
47:
48: int main() {
49:     sf::RenderWindow window(sf::VideoMode(300, 200),
50:         "SFML Plucked String Sound Lite");
51:     sf::Event event;
52:     char c;
53:     std::vector<std::unique_ptr<sf::Sound> > kSounds;
54:     std::vector<sf::Int16> samples;
55:     std::vector<std::vector<sf::Int16>> KSample;
56:     std::vector<std::unique_ptr<sf::SoundBuffer> > kBuffer;
57:     sf::Sound sound;
58:     sf::SoundBuffer buffer;
59:     std::string keys = "q2we4r5ty7u8i9op- [=zxdcfvgnbjmk,.;/' ";
60:
61:     auto func = [=] (int i) {
62:         std::vector<sf::Int16> samples;
63:         const double freq = 440.0 * pow(2.0, (i-24.0)/12.0);
64:         StringSound gsl(freq);
65:         samples = makeSamples(gsl);
```

```
66:         return samples;
67:     };
68:
69:     for (int i = 0; i < 37; i++) {
70:         KSample.push_back(func(i));
71:         // samples = KSample[i];
72:         // std::cout << samples.size() << std::endl;
73:     }
74:
75:     for (size_t i = 0; i < KSample.size(); i++) {
76:         if (!buffer.loadFromSamples(&(KSample[i].at(i)), KSample[i].size(
),
77:         2, SAMPLES_PER_SEC))
78:             throw std::runtime_error(
79:                 "sf::SoundBuffer: failed to load from samples.");
80:         kBuffer.push_back(std::make_unique<sf::SoundBuffer>(buffer));
81:     }
82:
83:     for (size_t i = 0; i < kBuffer.size(); i++) {
84:         sound.setBuffer(*kBuffer[i]);
85:         kSounds.push_back(std::make_unique<sf::Sound>(sound));
86:     }
87:
88:     /* freq = CONCERT_A * pow(2, 3.0/12.0);
89:     StringSound gs2(freq);
90:     sf::Sound sound2;
91:     sf::SoundBuffer buf2;
92:     samples = makeSamples(gs2);
93:     std::cout << samples.size() << std::endl;
94:     if (!buf2.loadFromSamples(&samples[0], samples.size(), 2, SAMPLES_PER
_SEC))
95:         throw std::runtime_error(
96:             "sf::SoundBuffer: failed to load from samples.");
97:     sound2.setBuffer(buf2); */
98:     int j = 0;
99:     while (window.isOpen()) {
100:         while (window.pollEvent(event)) {
101:             switch (event.type) {
102:                 case sf::Event::Closed:
103:                     window.close();
104:                     break;
105:
106:                 case sf::Event::TextEntered:
107:                     /*switch (event.key.code) {
108:                         case sf::Keyboard::A:
109:                             // sound1.play();
110:                             break;
111:                         case sf::Keyboard::C:
112:                             // sound2.play();
113:                             break;
114:                         default:
115:                             break;
116:                     }*/
117:                     c = static_cast<char>(event.text.unicode);
118:                     while (j < static_cast<int>(keys.size()) && c != keys[j])
119:                         j++;
120:                     }
121:                     if (j == static_cast<int>(keys.size())) {
122:                         throw std::runtime_error("wrong keys");
123:                     }
124:                     kSounds[j]->play();
125:                     j = 0;
126:                     break;
127:
```



```
128:         default:
129:             break;
130:     }
131:
132:     window.clear();
133:     window.display();
134: }
135: }
136: return 0;
137: }
```

```
1: // Copyright 2022 Anson Cheang
2: #ifndef _HOME_IIFORCE_BADNAME_COMP4_PS4B_STRINGSOUND_H_
3: #define _HOME_IIFORCE_BADNAME_COMP4_PS4B_STRINGSOUND_H_
4:
5: #include "CircularBuffer.h"
6: #include <stdint.h>
7: #include <iostream>
8: #include <cstdlib>
9: #include <vector>
10:
11: #include <SFML/Graphics.hpp>
12: #include <SFML/System.hpp>
13: #include <SFML/Audio.hpp>
14: #include <SFML/Window.hpp>
15:
16: class StringSound {
17: public:
18:     explicit StringSound(double frequency);
19:     explicit StringSound(std::vector <sf::Int16> init);
20:     StringSound (const StringSound &obj) = delete; // no copy const
21:     ~StringSound();
22:     void pluck();
23:     void tic();
24:     sf::Int16 sample();
25:     int time();
26: private:
27:     CircularBuffer * _cb;
28:     int _time;
29: };
30:
31: #endif // _HOME_IIFORCE_BADNAME_COMP4_PS4B_STRINGSOUND_H_
```

```
1: // Copyright 2022 Anson Cheang
2:
3: // This class is used to create the sound used by KSGuitarSim.cpp
4:
5: #include "StringSound.h"
6: #include <time.h>
7: #include <iostream>
8: #include <cmath>
9: #include <vector>
10: #include <iterator>
11: #include <random>
12: #include <chrono> //NOLINT
13:
14: #define SAMPLING_RATE 44100
15:
16: StringSound::StringSound(double frequency) {
17:     if (frequency <= 0) {
18:         throw std::invalid_argument(
19:             "StringSound(double frequency) : cant accept values of 0 or 1
ower");
20:     }
21:     size_t _size = ceil(SAMPLING_RATE/frequency);
22:     _cb = new CircularBuffer(_size);
23:     _time = 0;
24: }
25:
26: StringSound::StringSound(std::vector <sf::Int16> init) {
27:     _cb = new CircularBuffer(init.size());
28:     std::vector<sf::Int16>::iterator p;
29:     p = init.begin();
30:     while (p != init.end()) {
31:         _cb->enqueue(*p);
32:         p++;
33:     }
34:     _time = 0;
35: }
36:
37: StringSound::~StringSound() {
38:     delete _cb;
39:     _cb = nullptr;
40: }
41:
42: void StringSound::pluck() {
43:     if (_cb->size() <= 0) {
44:         while (!_cb->isEmpty()) {
45:             _cb->dequeue();
46:         }
47:     }
48:
49:     unsigned int secret = std::chrono::system_clock::now().time_since_epoch().count(); //NOLINT
50:     std::mt19937 gen(secret);
51:     while (!_cb->isFull()) {
52:         // generate random number
53:         std::uniform_int_distribution<int16_t> dist(-32768, 32767);
54:         _cb->enqueue(dist(gen));
55:     }
56: }
57:
58: void StringSound::tic() {
59:     if (!_cb->isFull()) {
60:         throw std::runtime_error("Tic() : The list isn't full");
61:     }
62:     // size_t size = _cb->size();
63:     sf::Int16 val1 = _cb->dequeue();
```

```
64:     sf::Int16 val2 = _cb->peek();
65:     sf::Int16 new_val = .996 * ((val1 + val2) / 2);
66:
67:     _cb->enqueue(new_val);
68:
69:     _time++;
70: }
71:
72: sf::Int16 StringSound::sample() {
73:     if (_cb->isEmpty()) {
74:         throw std::runtime_error("sample() : _cb is empty");
75:     }
76:     sf::Int16 sample = _cb->peek();
77:
78:     return sample;
79: }
80:
81: int StringSound::time() {
82:     return _time;
83: }
```

```
1: // Copyright 2022 Anson Cheang
2: #ifndef _HOME_IIFORCE_BADNAME_COMP4_PS4B_CIRCULARBUFFER_H_
3: #define _HOME_IIFORCE_BADNAME_COMP4_PS4B_CIRCULARBUFFER_H_
4:
5: #include <stdint.h>
6: #include <cstdlib>
7: #include <deque>
8:
9: class CircularBuffer {
10: public:
11:     CircularBuffer(size_t capacity); // create an empty ring buffer,
12:     // with given max capacity
13:     size_t size(); // return number of items currently in the buffer
14:     bool isEmpty(); // is the buffer empty (size equals zero)?
15:     bool isFull(); // is the buffer full (size equals capacity)?
16:     void enqueue(int16_t x); // add item x to the end
17:     int16_t dequeue(); // delete and return item from the front
18:     int16_t peek(); // return (but do not delete) item from the front
19:     unsigned int getCap();
20:
21: private:
22:     std::deque<int16_t> list;
23:     size_t currentSize;
24:     unsigned int maxCapacity;
25: };
26:
27: #endif // _HOME_IIFORCE_BADNAME_COMP4_PS4B_CIRCULARBUFFER_H_
```

```
1: // Copyright 2022 Anson Cheang
2:
3: // This is used to make the storage for the buffer.
4:
5: #include "CircularBuffer.h"
6: #include <iostream>
7: #include <string>
8:
9: CircularBuffer::CircularBuffer(size_t capacity) {
10:     std::string message =
11:         "CircularBuffer constructor: capacity must be greater than zero";
12:     if (capacity < 1) {
13:         throw std::invalid_argument(message);
14:     }
15:     currentSize = 0;
16:     maxCapacity = capacity;
17: }
18:
19: size_t CircularBuffer::size() {
20:     return list.size();
21: }
22:
23: bool CircularBuffer::isEmpty() {
24:     return list.size() <= 0;
25: }
26:
27: bool CircularBuffer::isFull() {
28:     return list.size() == maxCapacity;
29: }
30:
31: void CircularBuffer::enqueue(int16_t x) {
32:     if (isFull()) {
33:         throw std::runtime_error("enqueue: can't enqueue to a full ring s
34:     }
35:     list.push_back(x);
36: }
37:
38: int16_t CircularBuffer::dequeue() {
39:     if (isEmpty()) {
40:         throw std::runtime_error("dequeue: can't dequeue an empty ring");
41:     }
42:     int16_t val = list.front();
43:     list.pop_front();
44:     return val;
45: }
46:
47: int16_t CircularBuffer::peek() {
48:     if (isEmpty()) {
49:         throw std::runtime_error("peek: can't peek an empty ring");
50:     }
51:     return list.front();
52: }
53:
54: unsigned int CircularBuffer::getCap() {
55:     return maxCapacity;
56: }
```

```
1: // Copyright 2022 Anson Cheang
2: // test.cpp
3: // updated 1/30/22-1/31/22
4: // updated by Anson Cheang
5:
6: #define BOOST_TEST_DYN_LINK
7: #define BOOST_TEST_MODULE Main
8:
9: #include "CircularBuffer.h"
10: #include "StringSound.h"
11:
12: #include <iostream>
13: #include <sstream>
14: #include <string>
15: #include <boost/test/unit_test.hpp>
16:
17: BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
18:     BOOST_REQUIRE_THROW(StringSound l(0), std::invalid_argument);
19:     BOOST_REQUIRE_THROW(StringSound l2(-1), std::invalid_argument);
20:     BOOST_REQUIRE_NO_THROW(StringSound l3(5));
21:     StringSound l3(5);
22:     BOOST_REQUIRE_THROW(l3.tic(), std::runtime_error);
23:     BOOST_REQUIRE_THROW(l3.sample(), std::runtime_error);
24:     l3.pluck();
25:     BOOST_REQUIRE_NO_THROW(l3.tic());
26:     BOOST_REQUIRE_NO_THROW(l3.sample());
27: }
```

## PS5: DNA Alignment

**Overview:** The main point of this assignment is to take in a file with 2 lines of text, the length doesn't matter, and align them so that the edit distance is low. So for every mismatch it +1, an empty space/- is a +2, and a match is a +1. Afterward we needed to print out the lowest possible edit distance first, then the properly aligned dna sequence, along with the cost of the combination along with it, and finally an execution time, which is how long it ran. This was done on a matrix following something similar to the one shown on the dynamic programming slide by Needleman and Wunsch.

**Learned:** Learned how programming with a partner is supposed to go with 1 being the driver, and the other being the coder, essentially a lead and the listener, and after an hour or so we switch essentially allowing new views on old code.

**Algorithm/Data Structures/OO Design:** a vector of vector, in which we implement both line of text into a matrix and from there, it does its calculations and would place the numbers within the matrix to allow it to find the least edit distance, which is in the top right, and find its pathing for it to print out the alignment

**Difficulties:** The most difficult part of it was figuring out how we are supposed to do this assignment in the first place, although thanks to the class discord and my partner Andy we figured out that the pdf version has a strategy in how to do this, and as such we used that strategy albeit a bit edited. In this I learned to use every resource provided no matter how small the hint may be, or how I should probably review class slides after class.

### **Output:**

```
iiforce-badname@DESKTOP-NI0N3SH:~/Comp4/PS5$ ./EDistance < example10.txt
Edit Distance = 7
A T 1
A A 0
C - 2
A A 0
G G 0
T G 1
T T 0
A - 2
C C 0
C A 1
Execution time is 0.000697 seconds
```

### **Code:**



```
1: CC = g++
2: CFLAGS = -Wall -Werror -pedantic --std=c++14
3: LIBS = -lboost_unit_test_framework
4: DEPS = EDistance.h
5: SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
6:
7: %.o: %.cpp $(DEPS)
8:     $(CC) $(CFLAGS) -c $<
9:
10: all: EDistance
11:
12: EDistance: main.o EDistance.o
13:     $(CC) -g $(CFLAGS) -o EDistance $^ $(LIBS) $(SFMLFLAGS)
14:     cpplint --filter=--runtime/references *.cpp *.h
15:
16: clean:
17:     rm *.o EDistance
```

```
1: // Copyright 2022 Anson Cheang, and Andy Nguyen
2: #include <iostream>
3: #include <SFML/System.hpp>
4: #include "EDistance.h"
5:
6: using namespace std; //NOLINT
7:
8: int main(int argc, const char* argv[]) {
9:     sf::Clock clock;
10:    sf::Time t;
11:
12:    string input1, input2;
13:    cin >> input1;
14:    cin >> input2;
15:    EDistance dna(input1, input2);
16:    int distance = dna.optDistance();
17:    cout << "Edit Distance = " << distance << endl;
18:    cout << dna.alignment();
19:
20:    t = clock.getElapsedTime();
21:    // cout << "Edit Distance = " << distance << endl;
22:    cout << "Execution time is " << t.asSeconds() << " seconds \n";
23: }
```

```
1: // Copyright 2022 Anson Cheang, and Andy Nguyen
2: #ifndef _HOME_IIFORCE_BADNAME_COMP4_PS5_EDISTANCE_H_
3: #define _HOME_IIFORCE_BADNAME_COMP4_PS5_EDISTANCE_H_
4:
5: #include <iostream>
6: #include <string>
7: #include <vector>
8: #include <algorithm>
9:
10: using namespace std; //NOLINT
11:
12: class EDistance {
13: public:
14:     /* accepts the two strings to be compared, and allocates any
15:        data structures necessary into order to do the work
16:        (e.g., the NÃ\227M matrix).*/
17:     explicit EDistance(string N, string M);
18:
19:     /* returns the penalty for aligning chars a and b
20:        (this will be a 0 or 1)*/
21:     static int penalty(char a, char b);
22:
23:     // returns the minimum of the three arguments
24:     static int min(int a, int b, int c);
25:
26:     /* populates the matrix based on having the two strings, and returns
27:        the optimal distance (from the [0][0] cell of the matrix when done).
*/
28:     int optDistance();
29:
30:     /* traces the matrix and returns a string that can be printed to disp
lay
31:        the actual alignment. In general, this will be a multi-line string â
\200\224 i.e.,
32:        with embedded \n's.*/
33:     string alignment();
34:
35: private:
36:     string n, m;
37:     vector<vector<int>> matrix;
38: };
39:
40: #endif // _HOME_IIFORCE_BADNAME_COMP4_PS5_EDISTANCE_H_
```

```
1: // Copyright 2022 Anson Cheang, and Andy Nguyen
2:
3: #include <iostream>
4: #include <algorithm>
5: #include <cmath>
6: #include "EDistance.h"
7:
8: using namespace std; //NOLINT
9:
10: /* accepts the two strings to be compared, and allocates any
11:  data structures necessary into order to do the work
12:  (e.g., the NÅ\227M matrix).*/
13: EDistance::EDistance(string input_N, string input_M) {
14:     // n = row, m = column
15:     n = input_N;
16:     m = input_M;
17: }
18:
19: /* returns the penalty for aligning chars a and b
20:  (this will be a 0 or 1)*/
21: int EDistance::penalty(char a, char b) {
22:     if (a == b) {
23:         return 0;
24:     } else {
25:         return 1;
26:     }
27: }
28:
29: // returns the minimum of the three arguments
30: int EDistance::min(int a, int b, int c) {
31:     return std::min({a, b, c});
32: }
33:
34: /* populates the matrix based on having the two strings, and returns
35:  the optimal distance (from the [0][0] cell of the matrix when done).*/
36: int EDistance::optDistance() {
37:     int r = n.length();
38:     int c = m.length();
39:     int indel = 2;
40:     int match, del, insert;
41:
42:     for (int i = 0; i <= c; i++) {
43:         vector<int> temp;
44:         matrix.push_back(temp);
45:
46:         for (int j = 0; j <= r; j++) {
47:             matrix.at(i).push_back(0);
48:         }
49:     }
50:
51:     for (int i = 0; i <= c; i++) {
52:         matrix.at(i).at(r) = (c - i) * indel;
53:     }
54:
55:     for (int i = 0; i <= r; i++) {
56:         matrix.at(c).at(i) = (r - i) * indel;
57:     }
58:
59:     for (int i = c - 1; i >= 0; i--) {
60:         for (int j = r - 1; j >= 0; j--) {
61:             match = matrix.at(i + 1).at(j + 1) + penalty(m.at(i), n.at(j)
);
62:             del = matrix.at(i + 1).at(j) + indel;
63:             insert = matrix.at(i).at(j + 1) + indel;
64:             matrix.at(i).at(j) = min(match, del, insert);
```

```
65:         }
66:     }
67:     return matrix.at(0).at(0);
68: }
69:
70: /* traces the matrix and returns a string that can be printed to display
71: the actual alignment. In general, this will be a multi-line string â\200
\224 i.e.,
72: with embedded \n's.*/
73: string EDistance::alignment() {
74:     int indel = 2;
75:     string retStr;
76:
77:     int i = 0;
78:     int j = 0;
79:     int r = n.length();
80:     int c = m.length();
81:
82:     while (i < c || j < r) {
83:         if (i < c && j < r && matrix.at(i).at(j) == matrix.at(i + 1).at(j)
+ 1) + penalty(m[i], n[j])) { //NOLINT
84:             retStr = retStr + n[j] + " " + m[i] + " " + to_string(penalty
(m[i], n[j])) + "\n"; //NOLINT
85:             i++;
86:             j++;
87:         } else if (i < c && matrix.at(i).at(j) == matrix.at(i + 1).at(j)
+ indel) { //NOLINT
88:             retStr = retStr + "-" + " " + m[i] + " " + "2" + "\n";
89:             i++;
90:         } else if (j < r && matrix.at(i).at(j) == matrix.at(i).at(j + 1)
+ indel) { //NOLINT
91:             retStr = retStr + n[j] + " " + "-" + " " + "2" + "\n";
92:             j++;
93:         }
94:     }
95:     return retStr;
96: }
```

## PS6: Random Writer

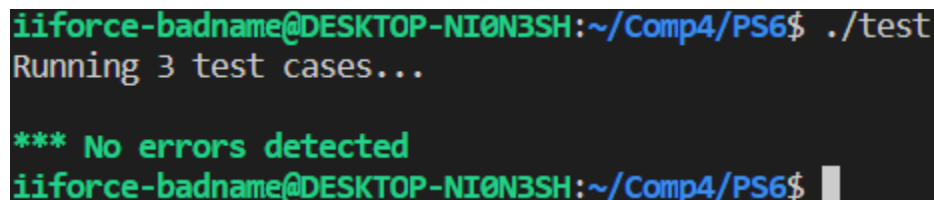
**Overview:** So this assignment has us make essentially a text scrambler using Markov model which is mostly done. The only part I didnt finish/start is the TextWriter.cpp, which is the main runner code, but I finished the support file, and test file. But continuing on, it is supposed to make a text scrambler using an input of numbers. It would then go through the list of possible combinations, use them to create the random string output and outputs them to the text.

**Learned:** the markov model, and how a textscramber is supposed to work. I also learned of a null character \0

**Algorithm/Data Structures/OO Design:** It contains a map of a string, map of a char, int. This is mainly used as storage for usage and editing which allows the code to function as it does as it stores the string, essentially every substring, then the char is every possible character after the string, and the int is the amount of it there it.

**Difficulties:** honestly this assignment was difficult to figure out, and I didnt finish on time. Like I figured out what I was doing thx to James Daly and the class discord, but I didnt have enough time to program in everything for me to finish it. So I did partially overcome it, by finishing the support file, but the main runner file is incomplete, as I didnt have enough time to do it.

**Output:**

A terminal window with a dark background and green text. The prompt is 'iiforce-badname@DESKTOP-NI0N3SH:~/Comp4/PS6\$'. The user has entered './test'. The output shows 'Running 3 test cases...' followed by '\*\*\* No errors detected' on a new line. The prompt is shown again at the bottom.

```
iiforce-badname@DESKTOP-NI0N3SH:~/Comp4/PS6$ ./test
Running 3 test cases...

*** No errors detected
iiforce-badname@DESKTOP-NI0N3SH:~/Comp4/PS6$
```

**Status:** Finished RandWriter.cpp, Did not finish the TextWriter.cpp, and as such there really is no output worth mentioning since there isn't any output in the first place.

**Code:**

```
1: CC = g++
2: CFLAGS = -Wall -Werror -pedantic --std=c++14
3: LIBS = -lboost_unit_test_framework
4: DEPS = RandWriter.h
5: SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
6:
7: %.o: %.cpp $(DEPS)
8:     $(CC) $(CFLAGS) -c $<
9:
10: all: test
11:
12: test: test.o RandWriter.o
13:     $(CC) -g $(CFLAGS) -o test $^ $(LIBS) $(SFMLFLAGS)
14:     cpplint --filter=--runtime/references *.cpp *.h
15:
16: clean:
17:     rm *.o test
```

```
1: // Copyright 2022 Anson Cheang
```



```
1: // Copyright 2022 Anson Cheang, and Andy Nguyen
2: #ifndef _HOME_IIFORCE_BADNAME_COMP4_PS6_RANDWRITER_H_
3: #define _HOME_IIFORCE_BADNAME_COMP4_PS6_RANDWRITER_H_
4:
5: #include <iostream>
6: #include <string>
7: #include <map>
8:
9: using namespace std; //NOLINT
10:
11: class RandWriter {
12: public:
13: // create a Markov model of order k from given text
14: RandWriter(string text, int k); // Assume that text has length at least
k.
15: // -----
16: int orderK() const; // order k of Markov model
17: // -----
18: // number of occurrences of kgram in text
19: int freq(string kgram) const; // throw an exception if kgram is not of
20: // length k
21: // -----
22: // number of times that character c follows kgram
23: // if order=0, return num of times char c appears
24: // (throw an exception if kgram is not of length k)
25: int freq(string kgram, char c) const;
26: // -----
27: // random character following given kgram
28: // (Throw an exception if kgram is not of length k.
29: // Throw an exception if no such kgram.)
30: char kRand(string kgram);
31: // -----
32: // generate a string of length L characters
33: // by simulating a trajectory through the corresponding
34: // Markov chain. The first k characters of the newly
35: // generated string should be the argument kgram.
36: // Throw an exception if kgram is not of length k.
37: // Assume that L is at least k.
38: string generate(string kgram, int L);
39: // -----
40: friend ostream& operator<<(ostream& out, RandWriter& markov);
41: // overload the stream insertion operator and display
42: // the internal state of the Markov Model. Print out
43: // the order, the alphabet, and the frequencies of
44: // the k-grams and k+1-grams.
45:
46: private:
47: string alphabet; //NOLINT
48: int order;
49: map<string, map<char, int>>> RM;
50: };
51:
52: ostream& operator<<(ostream& out, RandWriter& markov);
53:
54: #endif // _HOME_IIFORCE_BADNAME_COMP4_PS6_RANDWRITER_H_
```

```
1: // Copyright 2022 Anson Cheang
2: #include "RandWriter.h"
3: #include <utility>
4:
5: using namespace std; // NOLINT
6:
7: // -----
8: // create a Markov model of order k from given text
9: // Assume that text has length at least k.
10: RandWriter::RandWriter(string text, int k) {
11:     order = k;
12:     string storage = text;
13:     int length = text.length();
14:     int count = 0, l;
15:     char temp;
16:     string stringTemp, size;
17:     bool isThere;
18:
19:     for (int i = 0; i < order; i++) {
20:         storage = storage + text[i];
21:     }
22:     for (int i = 0; i < length; i++) {
23:         temp = text.at(i);
24:         isThere = false;
25:         l = alphabet.length();
26:         for (int j = 0; j < l; j++) {
27:             if (alphabet.at(j) == temp) {
28:                 isThere = true;
29:             }
30:         }
31:         if (!isThere) {
32:             alphabet = alphabet + temp;
33:         }
34:     }
35:     for (int i = 0; i < length; i++) {
36:         stringTemp.clear();
37:         stringTemp = storage.substr(i, order);
38:         RM.insert(make_pair(stringTemp, map<char, int>()));
39:         RM.at(stringTemp).insert(make_pair('\0', 0));
40:     }
41:
42:     char placement;
43:     for (int i = 0; i < length; i++) {
44:         stringTemp.clear();
45:         stringTemp = storage.substr(i, order+1);
46:         placement = stringTemp[order];
47:         stringTemp.clear();
48:         stringTemp = storage.substr(i, order);
49:         RM.at(stringTemp).insert(make_pair(placement, 0));
50:     }
51:     map<char, int>::iterator p;
52:
53:     for (int j = 0; j < length; j++) {
54:         stringTemp.clear();
55:         stringTemp = storage.substr(j, order);
56:         p = RM.at(stringTemp).find('\0');
57:         count = p->second;
58:         count++;
59:         RM.at(stringTemp).at('\0') = count;
60:     }
61:     for (int j = 0; j < length; j++) {
62:         stringTemp.clear();
63:         stringTemp = storage.substr(j, order+1);
64:         placement = stringTemp[order];
65:         stringTemp.clear();
```

```
66:         stringTemp = storage.substr(j, order);
67:         p = RM.at(stringTemp).find(placement);
68:         count = p->second;
69:         count++;
70:         RM.at(stringTemp).at(placement) = count;
71:     }
72: }
73: // -----
74: // order k of Markov model
75: int RandWriter::orderK() const {
76:     return order;
77: }
78: // -----
79: // number of occurrences of kgram in text
80: // throw an exception if kgram is not of
81: // length k
82: int RandWriter::freq(string kgram) const {
83:     if (kgram.length() != static_cast<unsigned int> (order)) {
84:         throw runtime_error("freq(string):kgram is not of length k(order)
");
85:     }
86:
87:     map<char, int>::const_iterator p;
88:
89:     p = RM.at(kgram).find('\0');
90:
91:     return p->second;
92: }
93: // -----
94: // number of times that character c follows kgram
95: // if order=0, return num of times char c appears
96: // (throw an exception if kgram is not of length k)
97: int RandWriter::freq(string kgram, char c) const {
98:     if (kgram.length() != static_cast<unsigned int> (order)) {
99:         throw runtime_error(
100:             "freq(string, char):kgram is not of length k(order)");
101:     }
102:
103:     string sub = kgram + c;
104:
105:     map<char, int>::const_iterator p;
106:     p = RM.at(kgram).find(c);
107:     return p->second;
108: }
109: // -----
110: // random character following given kgram
111: // (Throw an exception if kgram is not of length k.
112: // Throw an exception if no such kgram.)
113: char RandWriter::kRand(string kgram) {
114:     if (kgram.length() != static_cast<unsigned int> (order)) {
115:         throw runtime_error("kRand:kgram is not of length k(order)");
116:     }
117:     map<string, map<char, int>>::iterator t;
118:     map<char, int>::iterator p;
119:     t = RM.find(kgram);
120:     if (t == RM.end()) {
121:         throw runtime_error("kRand: could not find given kgram");
122:     }
123:     int amount = freq(kgram);
124:
125:     srand(static_cast<int>(time(NULL)));
126:     int rv = rand() % amount; //NOLINT
127:     p = RM.at(kgram).begin();
128:     p++;
129:     while (p != RM.at(kgram).end() && rv > p->second) {
```

```
130:         rv = rv - p->second;
131:     }
132:     return p->first;
133: }
134: // -----
135: // generate a string of length L characters
136: // by simulating a trajectory through the corresponding
137: // Markov chain. The first k characters of the newly
138: // generated string should be the argument kgram.
139: // Throw an exception if kgram is not of length k.
140: // Assume that L is at least k.
141: string RandWriter::generate(string kgram, int L) {
142:     if (kgram.length() != static_cast<unsigned int> (order)) {
143:         throw runtime_error("generate:kgram is not of length k(order)");
144:     }
145:     string returnString = kgram;
146:
147:     for (int i = 0; i < (L - order); i++) {
148:         returnString = returnString + kRand(returnString.substr(i, order)
);
149:     }
150:     return returnString;
151: }
152: // -----
153: // overload the stream insertion operator and display
154: // the internal state of the Markov Model. Print out
155: // the order, the alphabet, and the frequencies of
156: // the k-grams and k+1-grams.
157: ostream& operator<<(ostream& out, RandWriter& markov) {
158:     return out;
159: }
```

```
1: // Copyright 2022 Anson Cheang
2: #define BOOST_TEST_DYN_LINK
3: #define BOOST_TEST_MODULE Main
4:
5: #include "RandWriter.h"
6:
7: #include <iostream>
8: #include <sstream>
9: #include <string>
10: #include <boost/test/unit_test.hpp>
11:
12: BOOST_AUTO_TEST_CASE(order0) {
13:     BOOST_REQUIRE_NO_THROW(RandWriter("gagggagagggcgagaaa", 0));
14:
15:     RandWriter mm("gagggagagggcgagaaa", 0);
16:
17:     BOOST_REQUIRE(mm.orderK() == 0);
18:     BOOST_REQUIRE(mm.freq("") == 17);
19:     BOOST_REQUIRE_THROW(mm.freq("t"), std::runtime_error);
20:
21:     BOOST_REQUIRE(mm.freq("", 'g') == 9);
22:     BOOST_REQUIRE(mm.freq("", 'a') == 7);
23:     BOOST_REQUIRE(mm.freq("", 'c') == 1);
24:     BOOST_REQUIRE(mm.freq("", 'x') == 0);
25: }
26:
27: BOOST_AUTO_TEST_CASE(order1) {
28:     RandWriter mm("gagggagagggcgagaaa", 1);
29:
30:     BOOST_REQUIRE_THROW(mm.freq(""), std::runtime_error);
31:
32:     BOOST_REQUIRE(mm.freq("a") == 7);
33:     BOOST_REQUIRE(mm.freq("g") == 9);
34:     BOOST_REQUIRE(mm.freq("c") == 1);
35:
36:     BOOST_REQUIRE(mm.freq("a", 'a') == 2);
37:     BOOST_REQUIRE(mm.freq("a", 'c') == 0);
38:     BOOST_REQUIRE(mm.freq("a", 'g') == 5);
39:
40:     BOOST_REQUIRE_NO_THROW(mm.kRand("a"));
41:
42:     BOOST_REQUIRE_THROW(mm.kRand("t"), std::runtime_error);
43:
44:     BOOST_REQUIRE_THROW(mm.kRand("tt"), std::runtime_error);
45: }
46:
47: BOOST_AUTO_TEST_CASE(order2) {
48:     RandWriter mm("gagggagagggcgagaaa", 2);
49:
50:     BOOST_REQUIRE_NO_THROW(mm.freq("aa"));
51:     BOOST_REQUIRE_THROW(mm.freq("", 'g'), std::runtime_error);
52:
53:     BOOST_REQUIRE(mm.freq("aa") == 2);
54:     BOOST_REQUIRE(mm.freq("aa", 'a') == 1);
55: }
```

## PS7: Kronos Time Parsing

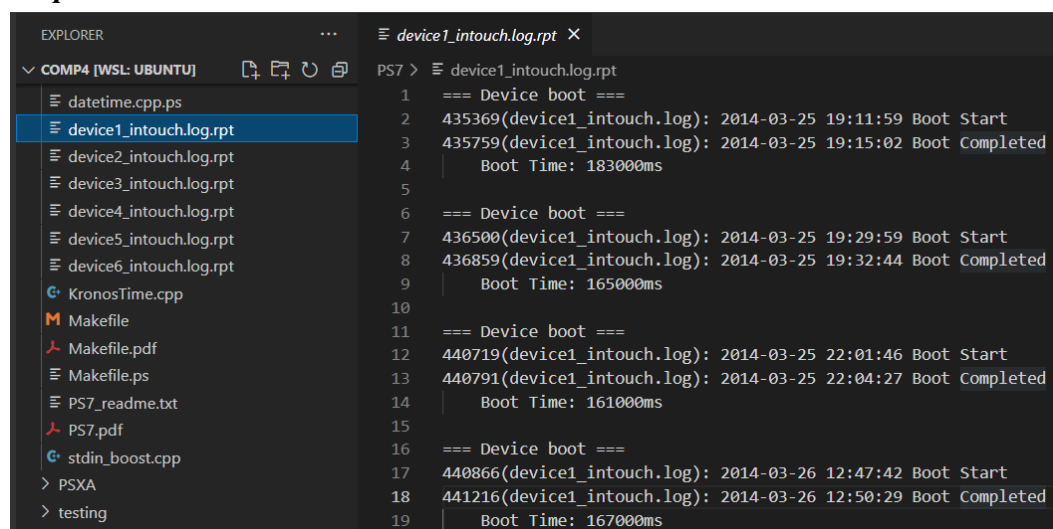
**Overview:** Essentially a way to read through the file, or rather in this case a log. So this assignment has us read from a log file, of extremely long length and size, in order to see if the device has booted, and if that boot has been completed or not. We also need the time it took to do the boot start and end, and as such, there is a boot time in order to see how long it took to finish the boot, or if it didn't finish you need to state whether or not it has finished yet. Then create a .log.rpt file containing all of it.

**Learned:** how to use regex, and the date and time class. Well the primary part of this assignment requires the use of regex and date\_time, since for datetime is used to say how long the boot took to finish, and regex is used to search through the log file, so kinda needed in order for it to be done

**Algorithm/Data Structures/OO Design:** There really isn't much within the program since it is a short program, but the key algorithm would most likely be my if statements, for searching for the beginning of the boot, when it starts, and when the boot ends or is finished, and would print things out appropriately, plus when it reads another boot begins would print out that the previous boot didn't finish. So it essentially allows us to know when the boot begins, completes, or has another boot begin.

**Difficulties:** The most difficult part of the assignment was trying to figure out what I was supposed to do in this assignment, I figured it out thanks to James Daly right before class when I brought up the point that I did not understand what I was doing. Didn't learn anything from it, but did reinforce the fact that if you don't understand an assignment ask others for insight. Another problem was regex\_match didn't work, so I replaced it with regex\_search and it works. Learned match is exact, and search is substring.

### **Output:**



```
1  === Device boot ===
2  435369(device1_intouch.log): 2014-03-25 19:11:59 Boot Start
3  435759(device1_intouch.log): 2014-03-25 19:15:02 Boot Completed
4  | Boot Time: 183000ms
5
6  === Device boot ===
7  436500(device1_intouch.log): 2014-03-25 19:29:59 Boot Start
8  436859(device1_intouch.log): 2014-03-25 19:32:44 Boot Completed
9  | Boot Time: 165000ms
10
11 === Device boot ===
12 440719(device1_intouch.log): 2014-03-25 22:01:46 Boot Start
13 440791(device1_intouch.log): 2014-03-25 22:04:27 Boot Completed
14 | Boot Time: 161000ms
15
16 === Device boot ===
17 440866(device1_intouch.log): 2014-03-26 12:47:42 Boot Start
18 441216(device1_intouch.log): 2014-03-26 12:50:29 Boot Completed
19 | Boot Time: 167000ms
```

### **Code:**

```
1: CC = g++
2: CFLAGS = -Wall -Werror -pedantic --std=c++14
3: LIBS = -lboost_unit_test_framework -lboost_date_time -lboost_regex
4: DEPS = RandWriter.h
5: SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
6:
7: %.o: %.cpp $(DEPS)
8:     $(CC) $(CFLAGS) -c $<
9:
10: all: ps7
11:
12: ps7: KronosTime.o
13:     $(CC) -g $(CFLAGS) -o ps7 $^ $(LIBS) $(SFMLFLAGS)
14:     cpplint --filter=--runtime/references *.cpp
15:
16: clean:
17:     rm *.o ps7
```

```
1: // date and time sample code
2: // Copyright (C) 2015 Fred Martin
3: // Tue Apr 21 17:37:46 2015
4:
5: // compile with
6: // g++ datetime.cpp -lboost_date_time
7: // Y. Rykalova 4/12/2021
8:
9: // http://www.boost.org/doc/libs/1_58_0/doc/html/date_time/gregorian.html
10: // http://www.boost.org/doc/libs/1_58_0/doc/html/date_time/posix_time.htm
```

1

```
11:
12: #include <iostream>
13: #include <string>
14: #include "boost/date_time/gregorian/gregorian.hpp"
15: #include "boost/date_time/posix_time/posix_time.hpp"
16:
17: using std::cout;
18: using std::cin;
19: using std::endl;
20: using std::string;
21:
22: using boost::gregorian::date;
23: using boost::gregorian::from_simple_string;
24: using boost::gregorian::date_period;
25: using boost::gregorian::date_duration;
26:
27: using boost::posix_time::ptime;
28: using boost::posix_time::time_duration;
29:
30: int main() {
31:     // Gregorian date stuff
32:     string s("2015-01-01");
33:     date d1(from_simple_string(s));
34:     date d2(2015, boost::gregorian::Apr, 21);
35:
36:     date_period dp(d1, d2); // d2 minus d1
37:
38:     date_duration dd = dp.length();
39:
40:     cout << "duration in days " << dd.days() << endl;
41:
42:     // Posix date stuff
43:     ptime t1(d1, time_duration(0, 0, 0, 0)); // hours, min, secs, nanosecs
44:     ptime t2(d2, time_duration(0, 0, 0, 0));
45:
46:     time_duration td = t2 - t1;
47:
48:     cout << "duration in hours " << td.hours() << endl;
49:     cout << "duration in ms " << td.total_milliseconds() << endl;
50: }
```