Computing IV

Project Portfolio

Erik van Tilborg

Spring 2022

Contents

1 PS0 - SFML Hello World Project

1.1 Overview

The overview of this project was to get a handle on how sfml works and to get it installed on our devices for future use. It was an introductory project to display a green circle and a sprite images in the sfml graphics window.

1.2 Key Concepts

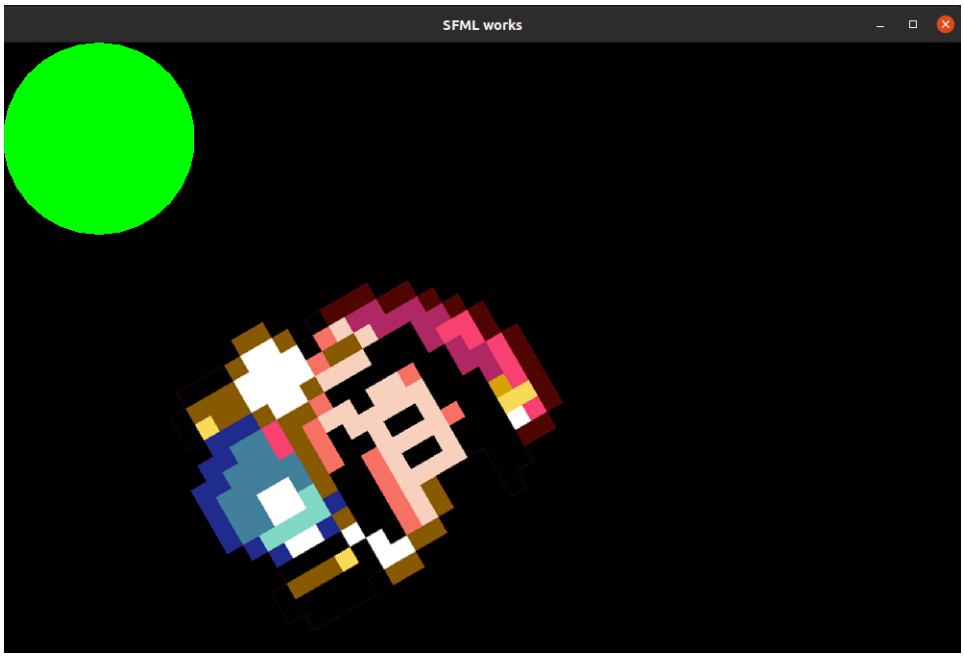
The key concepts for this assignment was getting acquainted with the workings of sfml in this intro project. To see how the window system works, how sprites work, and how things can interact with the keyboard when pressed.

1.3 Learnings

The learnings for this assignment was understanding how the window system works in sfml, getting a handle for how systems in sfml will work in the future projects in the course

1.4 Output

Figure 1: Program with green circle and sprite image



1.5 Code

1.5.1 Makefile

```
CXX =g++
CXXFLAGS = -Wall -Werror -pedantic --std=c++17
LIBS = -lsfml-graphics -lsfml-window -lsfml-system -
    lsfml-audio
LINTFILES = main.cpp
all: ps0 lint

%.o: %.cpp
    $(CXX) $(CXXFLAGS) -I/opt/homebrew/Cellar/sfml
       /2.5.1_1/include -c $<

ps0: main.o
    $(CXX) $(CXXFLAGS) -L/opt/homebrew/Cellar/sfml
       /2.5.1_1/lib -o $@ $^ $(LIBS)

lint: $(LINTFILES)
    cpplint $(LINTFILES)

clean:
    rm -f *.o
    rm -f ps0
    rm -f lint
```

1.5.2 main.cpp

```
// (C) Copyright Erik van Tilborg, 2022.
// Distributed under MIT license, available at
// https://opensource.org/licenses/MIT.

#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>
int main() {
    // Create the main window
    sf::RenderWindow window(sf::VideoMode(1000,
       1000), "SFML works");
    window.setFramerateLimit(60);
    sf::CircleShape shape(100.f);
```

```
12      shape.setFillColor(sf::Color::Green);
13      // Load a sprite to display
14      sf::Texture texture;
15      if (!texture.loadFromFile("sprite.png")) {
16          return EXIT_FAILURE;
17      }
18      sf::Sprite sprite(texture);
19      // Set sprite position
20      sprite.move(sf::Vector2f(500.f, 0.f));
21      // Start the game loop
22      while (window.isOpen()) {
23          // Process events
24          sf::Event event;
25          while (window.pollEvent(event)) {
26              // Close window: exit
27              if (event.type == sf::Event::Closed) {
28                  window.close();
29              }
30          }
31          // Clear screen
32          window.clear();
33          // Draw the circle
34          window.draw(shape);
35          // Draw the sprite
36          window.draw(sprite);
37          // Update the window
38          window.display();
39
40          if (sf::Keyboard::isKeyPressed(sf::Keyboard
                ::W)) {
41              sprite.move(0, -0.5);
42          }
43          if (sf::Keyboard::isKeyPressed(sf::Keyboard
                ::A)) {
44              sprite.move(-0.5, 0);
45          }
46          if (sf::Keyboard::isKeyPressed(sf::Keyboard
                ::S)) {
47              sprite.move(0, 0.5);
48          }
```

```
49          if (sf::Keyboard::isKeyPressed(sf::Keyboard
                ::D)) {
50              sprite.move(0.5, 0);
51          }
52          if (sf::Keyboard::isKeyPressed(sf::Keyboard
                ::Space)) {
53              sprite.rotate(15);
54          }
55      }
56      return EXIT_SUCCESS;
57  }
```

2 PS1 - Fibonacci LFSR and Image Encoding/Decoding

2.1 Overview

The project consisted of making a class that represented a linear feedback shift register. Once completed, the LFSR has a working implementation with 3 "tap" bits of its total 16 bits that get XORed together to form a new 16 bit binary sequence. This LFSR was then used in the process of both encoding and decoding images based on random seed values that were given to the LFSR. The colors of the pixels were controlled by the random values in the process, which resulted in a garbled image that could be reverted back by using the same seed as before.

2.2 Key Concepts

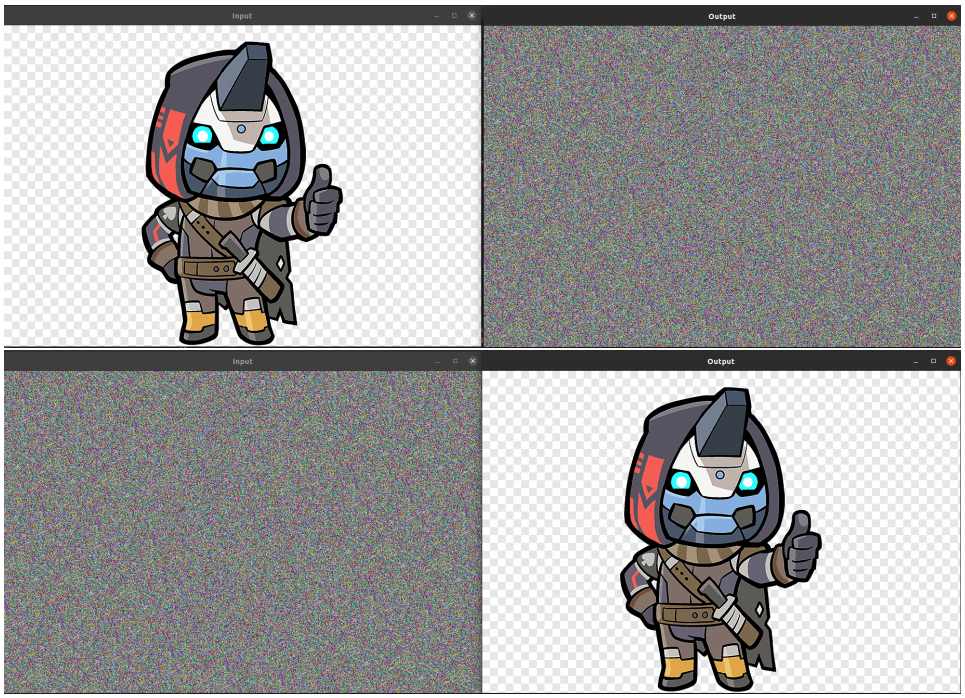
The key concepts from this assignment were understanding how an LFSR is implemented and the inner workings of how it actually works. Also this project served as an introduction to BOOST unit testing to verify that all aspects of written code is working as intended.

2.3 Learnings

The main takeaways that I had from this assignment was learning how an LFSR worked and what it may be used for. This was also the first time since Computing II that i did any sort of unit/function testing to verify that everything was working as it should be.

2.4 Output

Figure 2: Showing the encoding and decoding process



2.5 Code

2.5.1 Makefile

```
CXX = g++
CXXFLAGS = --std=c++11 --std=c++14 -Wall -Werror -
   pedantic
OBJECTS = FibLFSR.o test.o
TESTOBJ = FibLFSR.o main.o
LDLIBS = -lboost_unit_test_framework

all: ps1a main

ps1a: $(OBJECTS)
    $(CXX) $(CXXFLAGS) -o ps1a $(OBJECTS) $(LDLIBS)
FibLFSR.o: FibLFSR.cpp
    $(CXX) $(CXXFLAGS) -c FibLFSR.cpp -o FibLFSR.o
test.o: test.cpp
    $(CXX) $(CXXFLAGS) -c test.cpp -o test.o

main: $(TESTOBJ)
    $(CXX) $(CXXFLAGS) -o main $(TESTOBJ)
main.o: main.cpp
    $(CXX) $(CXXFLAGS) -c main.cpp -o main.o

clean:
    -rm ps1a main *.o
```

2.5.2 photomagic.cpp

```
#include <iostream>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "FibLFSR.h"

void transform(sf::Image& img, FibLFSR* obj);

int main(int argc, const char* argv[]){

    if(argc < 4){
```

```
12          std::cout << "Too Few Arguments: Requires 4
                arguments" << std::endl;
13          exit(1);
14      }
15
16      sf::Image img_in, img_out;
17
18      FibLFSR lfsr(argv[3]);
19
20      if(!img_in.loadFromFile(argv[1])){
21          return -1;
22      }
23
24      img_out = img_in;
25
26      transform(img_out, &lfsr);
27
28      sf::Vector2u size = img_in.getSize();
29      sf::RenderWindow window1(sf::VideoMode(size.x,
            size.y), "Input");
30      sf::RenderWindow window2(sf::VideoMode(size.x,
            size.y), "Output");
31
32      sf::Texture texture_in;
33      texture_in.loadFromImage(img_in);
34
35      sf::Texture texture_out;
36      texture_out.loadFromImage(img_out);
37
38      sf::Sprite sprite_in;
39      sprite_in.setTexture(texture_in);
40
41      sf::Sprite sprite_out;
42      sprite_out.setTexture(texture_out);
43
44      while (window1.isOpen() && window2.isOpen()) {
45          sf::Event event;
46          while (window1.pollEvent(event)) {
47              if (event.type == sf::Event::Closed){
48                  window1.close();
```

```
49              }
50          }
51
52          while (window2.pollEvent(event)) {
53              if (event.type == sf::Event::Closed){
54                  window2.close();
55              }
56          }
57
58          window1.clear(sf::Color::White);
59          window1.draw(sprite_in);
60          window1.display();
61          window2.clear(sf::Color::White);
62          window2.draw(sprite_out);
63          window2.display();
64      }
65
66      if (!img_out.saveToFile(argv[2])){
67          return -1;
68      }
69
70      return 0;
71  }
72
73  void transform(sf::Image& img, FibLFSR* obj){
74      sf::Color pixel;
75      for(size_t x = 0; x < img.getSize().x; x++){
76          for(size_t y = 0; y < img.getSize().y; y++){
77              pixel = img.getPixel(x,y);
78              pixel.r = pixel.r ^ obj->generate(8);
79              pixel.g = pixel.g ^ obj->generate(8);
80              pixel.b = pixel.b ^ obj->generate(8);
81              img.setPixel(x, y, pixel);
82          }
83      }
84  }
```

2.5.3 FibLFSR.h

```
1 #pragma once
```

```
2
3 #include <string>
4 #include <vector>
5 #include <iostream>
6
7 class FibLFSR {
8 public:
9     FibLFSR(const std::string& seed); // constructor
          to create LFSR with the given initial seed
10     int step(); // simulate one step and return the
         new bit as 0 or 1
11     int generate(int k); // simulate k steps and
         return k-bit integer
12     int getSize() const;
13     int getBitAtIndex(int i) const;
14 private:
15     std::vector<int> vecOfBits;
16 };
17
18 std::ostream& operator<<(std::ostream& out, const
     FibLFSR& obj);
```

2.5.4 FibLFSR.cpp

```
1 /*
     ************************************************************

2  *Name: <Erik van Tilborg>
3  *Course name: <COMP.2040>
4  *Assignment: <PS1a>
5  *Instructor's name: <Dr. James Daly>
6  *Date: <1/31/22>
7  *Sources Of Help: <class notes>
8 ************************************************************
     */
9 #include "FibLFSR.h"
10
11 #define MAXBITS 16 //LFSR designed to take in 16 bit
      strings, nothing more, nothing less
12
```

```
13 FibLFSR::FibLFSR(const std::string& seed){
14     if(seed.size() > MAXBITS || seed.size() <
          MAXBITS){
15         throw std::invalid_argument("Expected a 16
              bit string for the seed");
16     }
17     for(size_t i = 0; i < seed.size(); i++){
18         vecOfBits.push_back(seed[i] - 48);
19     }
20 }
21
22 int FibLFSR::step(){
23     int XORresult;
24     int vecSize = vecOfBits.size();
25     XORresult = vecOfBits[0] ^ vecOfBits[2]; //XOR
          bit 15 and 13
26     XORresult = XORresult ^ vecOfBits[3]; //XOR bit
          12 with previous result
27     XORresult = XORresult ^ vecOfBits[5]; //XOR bit
          10 with previous result
28     for (int i = 0; i < vecSize; i++){ //shift
          internal state
29         if((i + 1) > vecSize){
30             vecOfBits.push_back(0);
31         }
32         else{
33             vecOfBits[i] = vecOfBits[i + 1];
34         }
35     }
36     vecOfBits.pop_back();
37     vecOfBits.push_back(XORresult);
38     return XORresult; //return final XOR value
39 }
40
41 int FibLFSR::generate(int k) {
42     if (k > 32 || k < 0){ //check if k is out of
          range
43         std::cout << "The k value is out of range
              and creates an overflow issue... exiting
              program" << std::endl;
```

```
44          exit(1);
45      }
46      int val = 0; //int val = 0
47      for (int i = 0; i < k; i++){ //loop step for k
            times
48          val *= 2; //take val * 2
49          val = val + step(); //then add return val of
                step
50      }
51      return val; //return final value
52  }
53
54  int FibLFSR::getSize() const {
55      return vecOfBits.size();
56  }
57
58  int FibLFSR::getBitAtIndex(int i) const {
59      return vecOfBits[i];
60  }
61
62  std::ostream& operator << (std::ostream& out, const
        FibLFSR& obj){
63      for (int i = 0; i < obj.getSize(); i++) {
64          out << obj.getBitAtIndex(i);
65      }
66      return out;
67  }
```

2.5.5 initialTest.cpp

```
1  /*
       ************************************************************

2   *Name: <Erik van Tilborg>
3   *Course name: <COMP.2040>
4   *Assignment: <PS1a>
5   *Instructor's name: <Dr. James Daly>
6   *Date: <1/31/22>
7   *Sources Of Help: <class notes>
8  ************************************************************
```

```
    */
#include <iostream>
#include "FibLFSR.h"

int main(int argc, const char * argv[]){

    FibLFSR flfsr1("1011011000110110");
    int i = 0;

    while(i < 10){
        int x = flfsr1.step();
        std::cout << flfsr1 << " " << x << std::endl
            ;
        i++;
    }

    std::cout << "\n";

    FibLFSR flfsr2("1011011000110110");
    int j = 0;
    while(j < 7){
        int y = flfsr2.generate(5);
        std::cout << flfsr2 << " " << y << std::endl
            ;
        j++;
    }

    return 0;
}
```

2.5.6 test.cpp

```
// Initial tests provided by:
// Dr. Rykalova
// test.cpp for PS1a
// updated 1/31/2020

#include <iostream>
#include <string>

```

```
 9 #include "FibLFSR.h"
10
11 #define BOOST_TEST_DYN_LINK
12 #define BOOST_TEST_MODULE Main
13 #include <boost/test/unit_test.hpp>
14
15 BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
16
17   FibLFSR l("1011011000110110");
18   BOOST_REQUIRE(l.step() == 0);
19   BOOST_REQUIRE(l.step() == 0);
20   BOOST_REQUIRE(l.step() == 0);
21   BOOST_REQUIRE(l.step() == 1);
22   BOOST_REQUIRE(l.step() == 1);
23   BOOST_REQUIRE(l.step() == 0);
24   BOOST_REQUIRE(l.step() == 0);
25   BOOST_REQUIRE(l.step() == 1);
26
27   FibLFSR l2("1011011000110110");
28   BOOST_REQUIRE(l2.generate(9) == 51);
29 }
30
31 //beginning of added tests:
32
33 BOOST_AUTO_TEST_CASE(caseNot16BitString){
34 /**test to see if the constructor properly throws an
35  * invalid_argument if the seed given is longer or
36  * shorter than 16 bits
37  **/
38
39     BOOST_REQUIRE_THROW(FibLFSR Test2a("
          10001011011010100");, std::invalid_argument);
           //testing with 17 bit string
40     BOOST_REQUIRE_THROW(FibLFSR Test2b("
          101001111001100");, std::invalid_argument);
          //testing with 15 bit string
41
42 }
43
44 BOOST_AUTO_TEST_CASE(unnamedTest){
```

```
45 /**test to make sure the constructor is properly
46  * converting the information from the initial
47  * bit string into the correct position in the
48  * private vector object
49  **/
50
51     FibLFSR Test3("0011100010101100");
52     BOOST_REQUIRE(Test3.getBitAtIndex(0) == 0);
53     BOOST_REQUIRE(Test3.getBitAtIndex(1) == 0);
54     BOOST_REQUIRE(Test3.getBitAtIndex(2) == 1);
55     BOOST_REQUIRE(Test3.getBitAtIndex(3) == 1);
56     BOOST_REQUIRE(Test3.getBitAtIndex(4) == 1);
57     BOOST_REQUIRE(Test3.getBitAtIndex(5) == 0);
58     BOOST_REQUIRE(Test3.getBitAtIndex(6) == 0);
59     BOOST_REQUIRE(Test3.getBitAtIndex(7) == 0);
60     BOOST_REQUIRE(Test3.getBitAtIndex(8) == 1);
61     BOOST_REQUIRE(Test3.getBitAtIndex(9) == 0);
62     BOOST_REQUIRE(Test3.getBitAtIndex(10) == 1);
63     BOOST_REQUIRE(Test3.getBitAtIndex(11) == 0);
64     BOOST_REQUIRE(Test3.getBitAtIndex(12) == 1);
65     BOOST_REQUIRE(Test3.getBitAtIndex(13) == 1);
66     BOOST_REQUIRE(Test3.getBitAtIndex(14) == 0);
67     BOOST_REQUIRE(Test3.getBitAtIndex(15) == 0);
68
69 }
```

3 PS2 - NBody Physics Simulation

3.1 Overview

The goal of this assignment was to create a universe simulation with the first four planets in our solar system. With this, the planets would revolve around the sun based on some physics calculations, taking into account: velocity, position, mass, etc. This was all using SFML's window system and converting viewport coordinates to those of an xy plane or coordinate grid.

3.2 Key Concepts

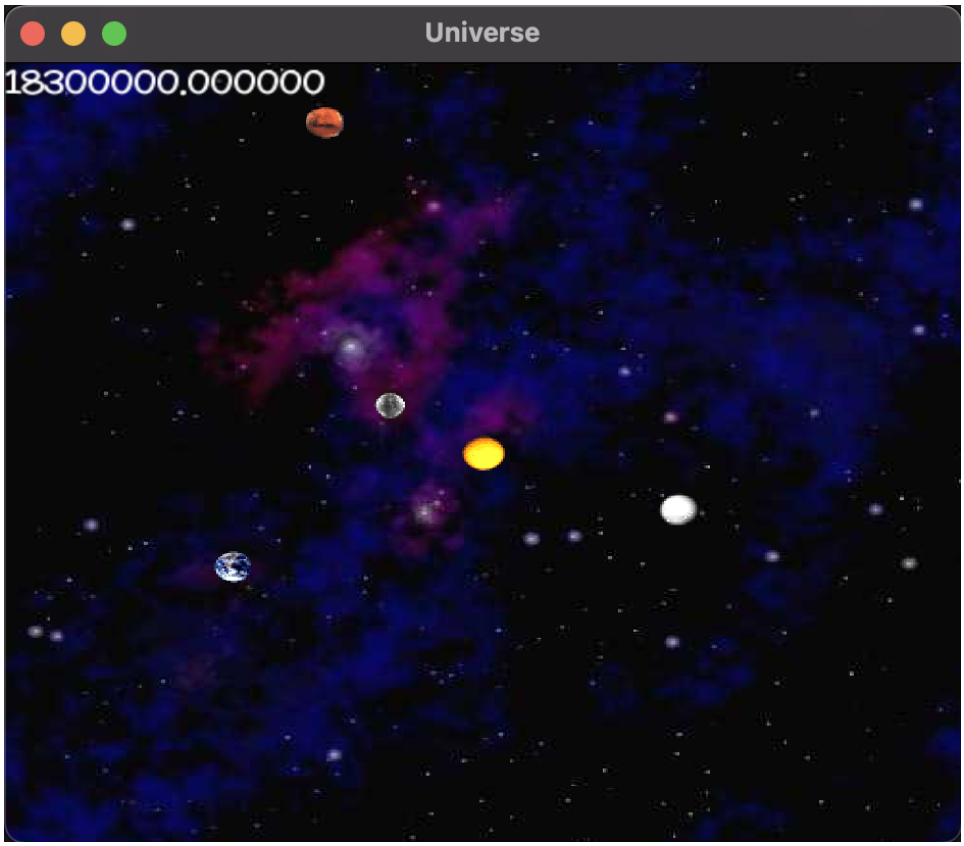
The key concepts for this assignment were creating multiple classes to house all of the necessary information and calculations needed to get the planets to correctly revolve around the sun. Another one was getting the calculations to work right with on another to make the sprites move in the correct direction on the viewport screen.

3.3 Learnings

The main takeaways that I had from this assignment was mainly to do with the interactions with the planets, physics calculations, and the viewport conversions. All of these together is what made the project possible. Getting the emulated "physics engine" to work with a couple of functions was definitely something that was a big takeaway from this assignment.

3.4 Output

Figure 3: Image of the program running



3.5 Code

3.5.1 Makefile

```
CXX = g++
CXXFLAGS = --std=c++17 -Wall -Werror -pedantic
OBJECTS = CelestialBody.o Universe.o main.o
SFML = -lsfml-graphics -lsfml-window -lsfml-system -
    lsfml-audio

all: NBody

NBody: $(OBJECTS)
    $(CXX) $(CXXFLAGS) -o NBody $(OBJECTS) $(SFML)
Universe.o: Universe.cpp
    $(CXX) $(CXXFLAGS) -c Universe.cpp -o Universe.o
CelestialBody.o: CelestialBody.cpp
    $(CXX) $(CXXFLAGS) -c CelestialBody.cpp -o
       CelestialBody.o
main.o: main.cpp
    $(CXX) $(CXXFLAGS) -c main.cpp -o main.o

clean:
    -rm NBody *.o
```

3.5.2 main.cpp

```
#include <iostream>
#include <string.h>
#include "SFML/System.hpp"
#include "SFML/Window.hpp"
#include "SFML/Graphics.hpp"
#include "SFML/Audio.hpp"
#include "Universe.h"
#include "CelestialBody.h"

int main(int argc, const char * argv[]){

    double totalTime = std::stod(argv[1]);
    double secondsPerStep = std::stod(argv[2]);
    Universe MilkyWay;
```

```
15      std::cin >> MilkyWay;
16
17      int windowSizeX = 500, windowSizeY = 500;
18
19      for(int i = 0; i < MilkyWay.getNumBodies(); i++)
           {
20          auto temp = MilkyWay.getPlanet(i);
21          temp->setWindowDimensions(windowSizeX,
               windowSizeY);
22          temp->setPosition(temp->getXPos(), temp->
               getYPos());
23      }
24
25      sf::Image backdropIMG;
26      if(!backdropIMG.loadFromFile("starfield.jpg")){
27          return -1;
28      }
29
30      sf::Texture backdropTEX;
31      backdropTEX.loadFromImage(backdropIMG);
32      backdropTEX.setRepeated(true);
33
34      sf::Sprite Backdrop;
35      Backdrop.setTexture(backdropTEX);
36
37      sf::Font font;
38      if(!font.loadFromFile("comic_sans.ttf")){
39          std::cout << "Could not open font file" <<
               std::endl;
40          return -1;
41      }
42
43      sf::Text text;
44      text.setFont(font);
45      text.setCharacterSize(20);
46      text.setFillColor(sf::Color::White);
47
48      sf::Music music;
49      if(MilkyWay.getNumBodies() <= 5){
50          if(!music.openFromFile("BigBang.wav")){
```

```
51              return -1;
52          }
53      } else{
54          if(!music.openFromFile("starwars.wav")){
55              return -1;
56          }
57      }
58
59      music.setLoop(false);
60      music.play();
61
62      sf::RenderWindow window(sf::VideoMode(
           windowSizeX, windowSizeY), "Universe");
63      double timeElapsed = 0;
64      window.setFramerateLimit(60);
65      while (window.isOpen()){
66          sf::Event event;
67          //while(totalTime > timeElapsed){
68              while(window.pollEvent(event)){
69                  if(event.type == sf::Event::Closed||
                        timeElapsed > totalTime){
70                      window.close();
71                  }
72              }
73              std::string Time = (std::to_string(
                   timeElapsed));
74              text.setString(Time);
75              window.clear();
76              window.draw(Backdrop);
77              window.draw(text);
78
79              for(int i = 0; i < MilkyWay.getNumBodies
                   (); i++){
80                  window.draw(*MilkyWay.getPlanet(i));
81              }
82              //MilkyWay.printContents();
83              window.display();
84              MilkyWay.step(secondsPerStep);
85              if(timeElapsed > totalTime){
86                  secondsPerStep = 0;
```

```
87              }
88              timeElapsed += secondsPerStep;
89          //}
90      }
91      std::cout << MilkyWay;
92
93      return 0;
94  }
```

3.5.3 CelestialBody.h

```
1  #pragma once
2
3  #include <iostream>
4  #include <string>
5  #include <SFML/Graphics.hpp>
6
7  class CelestialBody : public sf::Drawable {
8
9  public:
10 CelestialBody(double x, double y, double xVelo,
       double yVelo, double mass, const std::string&
       planetName);
11
12 double getXPos() const {
13     return xPos;
14 }
15
16 double getYPos() const {
17     return yPos;
18 }
19
20 sf::Vector2f getPos() const {
21     return sf::Vector2f (xPos, yPos);
22 }
23
24 double getXVelo() const {
25     return xVelocity;
26 }
27
```

```
28 double getYVelo() const {
29     return yVelocity;
30 }
31
32 void setVelo(double x, double y){
33     xVelocity = x;
34     yVelocity = y;
35 }
36
37 double getMass() const {
38     return Mass;
39 }
40
41 void setRadius(double rad){
42     radius = rad;
43 }
44
45 void setForce(double F){
46     force = F;
47 }
48
49 double getForce() const {
50     return force;
51 }
52
53 void setAccel(double accX, double accY){
54     accelX = accX;
55     accelY = accY;
56 }
57
58 double getAccelX() const {
59     return accelX;
60 }
61
62 double getAccelY() const {
63     return accelY;
64 }
65
66 void setPosition(double x, double y);
67
```

```
68 void setWindowDimensions(int width, int depth){
69     windowX = width;
70     windowY = depth;
71 }
72
73 std::string getFilename() const{
74     return PlanetName;
75 }
76
77 void setViewportCoords();
78
79 private:
80     int windowX, windowY;
81     double xPos, yPos, xVelocity, yVelocity, Mass,
          radius, force, accelX, accelY;
82     std::string PlanetName;
83     sf::Image planetIMG;
84     sf::Texture planetTEX;
85     sf::Sprite Planet;
86
87     virtual void draw(sf::RenderTarget &target, sf::
          RenderStates states) const {
88         target.draw(Planet, states);
89     }
90 };
```

3.5.4 CelestialBody.cpp

```
1 #include <iostream>
2 #include <string>
3 #include "CelestialBody.h"
4
5 CelestialBody::CelestialBody(double x, double y,
    double xVelo, double yVelo, double mass, const
    std::string& planetName){
6     xPos = x;
7     yPos = y;
8     xVelocity = xVelo;
9     yVelocity = yVelo;
10    Mass = mass;
```

```
11      PlanetName = planetName;
12      //setPos(xPos, yPos);
13      if(!planetIMG.loadFromFile(planetName)){
14          std::cout << "Could not load file" << std::
                endl;
15      }
16      planetTEX.loadFromImage(planetIMG);
17      Planet.setTexture(planetTEX);
18  }
19
20  void CelestialBody::setPosition (double x, double y)
        {
21      xPos = x;
22      yPos = y;
23      double viewportX = (x / radius) * (windowX / 2)
            + (windowX / 2);
24      double viewportY = (-y / radius) * (windowY / 2)
            + (windowY / 2);
25      double viewXPos = viewportX;
26      double viewYPos = viewportY;
27      Planet.setOrigin(Planet.getTexture()->getSize().
            x /2, Planet.getTexture()->getSize().y /2);
28      Planet.setPosition(viewXPos, viewYPos);
29  }
```

3.5.5 Universe.h

```
1  #pragma once
2  #include <iostream>
3  #include <iomanip>
4  #include <memory>
5  #include "CelestialBody.h"
6
7  class Universe {
8  public:
9      Universe() = default;
10
11     int getSize(){
12         return universeContents.size();
13     }
```

```
14
15      double getRadius() const {
16          return uniRadius;
17      }
18
19      int getNumBodies() const {
20          return numCelestialBodies;
21      }
22
23      std::shared_ptr<CelestialBody> getPlanet(int x){
24          return universeContents[x];
25      }
26
27      void printContents();
28
29      void step (double seconds);
30      double calcForce(CelestialBody& anchor,
            CelestialBody& source);
31      double getXDistance(CelestialBody& anchor,
            CelestialBody& source);
32      double getYDistance(CelestialBody& anchor,
            CelestialBody& source);
33      double getRDistance(CelestialBody& anchor,
            CelestialBody& source);
34
35
36      //sf::Vector2f normalize(const sf::Vector2f& vec
            );
37      //double getMagnitude(const sf::Vector2f& vec);
38
39      friend std::istream& operator>> (std::istream&
            in, Universe& uni);
40      friend std::ostream& operator<< (std::ostream&
            out, Universe& uni);
41
42  private:
43      int numCelestialBodies;
44      double uniRadius;
45      std::vector<std::shared_ptr<CelestialBody> >
            universeContents;
```

```
46 };
```

3.5.6 Universe.cpp

```
1 #include "Universe.h"
2 #include <cmath>
3 #define G 6.674e-11
4
5 std::istream& operator>> (std::istream& in, Universe
    & uni){
6     double x, y, xVelo, yVelo, mass;
7     std::string planetName;
8     in >> uni.numCelestialBodies;
9     in >> uni.uniRadius;
10    int i = 0;
11    while(i < uni.numCelestialBodies){
12        in >> x >> y >> xVelo >> yVelo >> mass >>
              planetName;
13        auto temp = std::make_shared<CelestialBody>(
              x, y, xVelo, yVelo, mass, planetName);
14        temp->setRadius(uni.uniRadius);
15        uni.universeContents.push_back(temp);
16        i++;
17    }
18
19    return in;
20 }
21
22 std::ostream& operator<< (std::ostream& out,
    Universe& uni){
23    out << uni.getNumBodies() << std::endl;
24    out << uni.getRadius() << std::endl;
25    for(int i = 0; i < uni.getNumBodies(); i++){
26        out << std::setw(14) << uni.getPlanet(i)->
              getXPos() << std::setw(14) << uni.
              getPlanet(i)->getYPos()
27        << std::setw(14) << uni.getPlanet(i)->
              getXVelo() << std::setw(14) << uni.
              getPlanet(i)->getYVelo()
28        << std::setw(14) << uni.getPlanet(i)->
```

```
            getMass() << std::setw(20) << uni.
            getPlanet(i)->getFilename() << std::endl;
29      }
30      return out;
31  }
32
33
34  double Universe::getXDistance(CelestialBody& anchor,
        CelestialBody& source){
35      return (source.getXPos() - anchor.getXPos());
36  }
37
38  double Universe::getYDistance(CelestialBody& anchor,
        CelestialBody& source){
39      return (source.getYPos() - anchor.getYPos());
40  }
41
42  double Universe::getRDistance(CelestialBody& anchor,
        CelestialBody& source){
43      double xDist, yDist, rDist;
44      xDist = getXDistance(anchor, source);
45      yDist = getYDistance(anchor, source);
46      rDist = sqrt((xDist * xDist) + (yDist * yDist));
47      return rDist;
48  }
49
50  double Universe::calcForce(CelestialBody& anchor,
       CelestialBody& source){
51      double rDist = getRDistance(anchor, source);
52      double force = ((G * ((anchor.getMass() * source
           .getMass())) / (rDist * rDist)));
53      return force;
54  }
55
56  void Universe::step (double seconds){
57      for (int i = 0; i < numCelestialBodies; i++){
58          double accelX, accelY, uptXVelo = 0,
               uptYVelo = 0, force, xDist, yDist, rDist,
                mass;
59          double initXVelo = getPlanet(i)->getXVelo(),
```

```
            initYVelo = getPlanet(i)->getYVelo(),
           forceX = 0, forceY = 0;
60          mass = getPlanet(i)->getMass();
61          for (int j = 0; j < numCelestialBodies; j++)
               {
62              if(j != i){
63                  force = calcForce(*getPlanet(i), *
                       getPlanet(j));
64                  xDist = getXDistance(*getPlanet(i),
                       *getPlanet(j));
65                  yDist = getYDistance(*getPlanet(i),
                       *getPlanet(j));
66                  rDist = getRDistance(*getPlanet(i),
                       *getPlanet(j));
67                  forceX += force * (xDist / rDist);
68                  forceY += force * (yDist / rDist);
69              }
70          }
71
72          accelX = forceX / mass;
73          accelY = forceY / mass;
74          getPlanet(i)->setAccel(accelX, accelY);
75
76          uptXVelo = initXVelo + (seconds * accelX);
77          uptYVelo = initYVelo + (seconds * accelY);
78          getPlanet(i)->setVelo(uptXVelo, uptYVelo);
79
80      }
81      for(int k = 0; k < numCelestialBodies; k++){
82          double newXV, newYV;
83          newXV = getPlanet(k)->getXPos() + (seconds *
                getPlanet(k)->getXVelo());
84          newYV = getPlanet(k)->getYPos() + (seconds *
                getPlanet(k)->getYVelo());
85          getPlanet(k)->setPosition(newXV, newYV);
86      }
87  }
```

4 PS3 - Recursive Graphics (Triangle Fractal)

4.1 Overview

This assignment was to create a program that recursively drew triangles to the screen using SFML graphics. This was based off of the Sierpinski Triangle method but is not one for one with the original.

4.2 Key Concepts

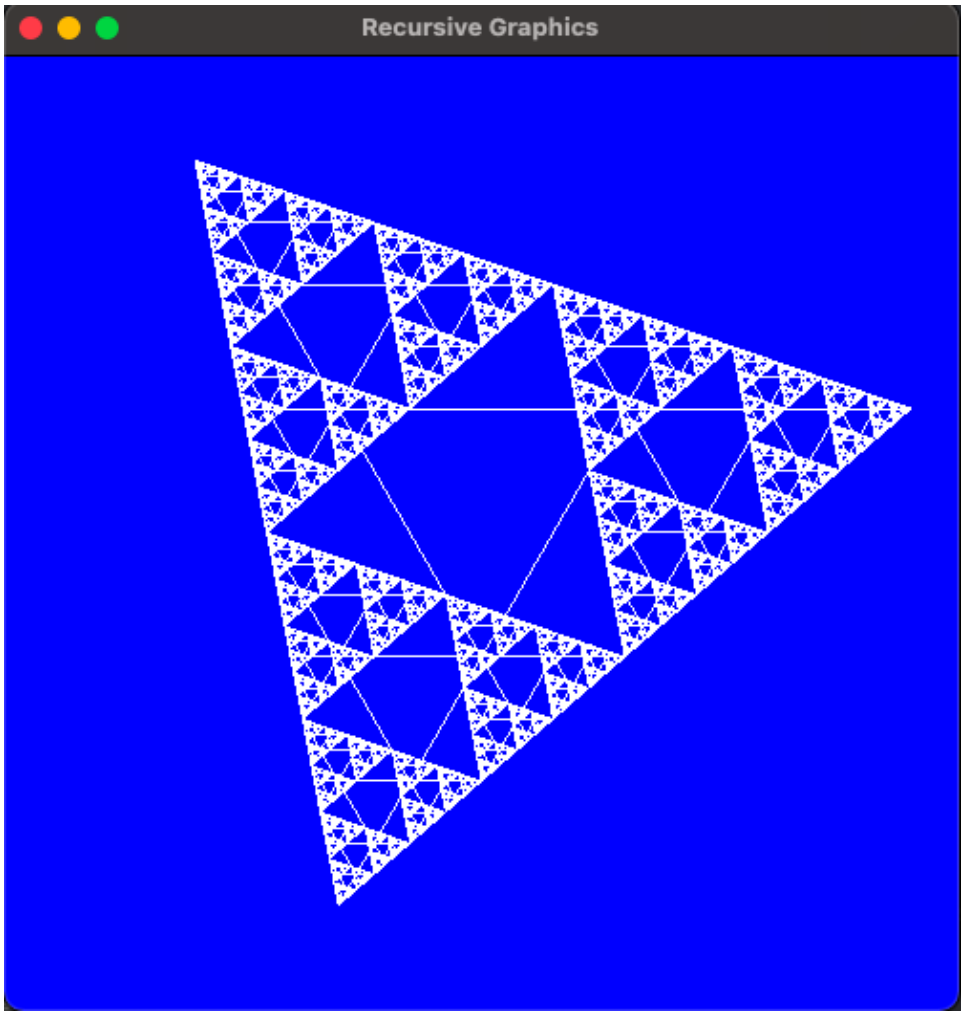
The key concepts for this assignment was mainly recursion. In my implementation I also used shared pointers for more ease of use as well as a vertex array from the sf class to house the points for the triangles.

4.3 Learnings

The main takeaways that I had from this assignment was learning how the implementation of smart pointers worked, as well as how they operated compared to raw pointers. it was also an introduction to the sf::VertexArray for me. It did make drawing the triangles a lot easier.

4.4 Output

Figure 4: caption here



4.5 Code

4.5.1 Makefile

```
CXX =g++
CXXFLAGS = -Wall -Werror -pedantic --std=c++17
LIBS = -lsfml-graphics -lsfml-window -lsfml-system -
    lsfml-audio
DEPS = Triangle.h
all: TFractal

%.o: %.cpp $(DEPS)
    $(CXX) $(CXXFLAGS) -I/opt/homebrew/Cellar/sfml
       /2.5.1_1/include -c $<

TFractal: TFractal.o Triangle.o
    $(CXX) $(CXXFLAGS) -L/opt/homebrew/Cellar/sfml
       /2.5.1_1/lib -o $@ $^ $(LIBS)

clean:
    rm -f *.o
    rm -f TFractal
```

4.5.2 TFractal.cpp

```
// (C) Copyright Erik van Tilborg, 2022.
// Distributed under MIT license, available at
// https://opensource.org/licenses/MIT.

#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>
#include "Triangle.h"

void fTree(std::vector<Triangle> *triangleHouse, sf
    ::Vector2<double>
centerPoint, double base, int recur, sf::
    PrimitiveType type);

int main(int argc, const char* argv[]) {
```

```
15      double windowSize = 500, baseLength = std::stod(
            argv[1]);
16      int recursions = std::stoi(argv[2]);
17
18      if (baseLength > windowSize / 2) {
19          windowSize *= 4;
20      }
21
22      sf::Vector2<double> centerPoint;
23      centerPoint.x = (windowSize / 2);
24      centerPoint.y = (windowSize / 2);
25
26      std::vector<Triangle> triangleStorage;
27      fTree(&triangleStorage, sf::Vector2<double>
28      (0.5 * windowSize, 0.5 * windowSize), baseLength
            , recursions,
29      sf::LineStrip);
30
31      sf::RenderWindow window(sf::VideoMode(windowSize
            , windowSize),
32      "Recursive Graphics");
33      window.setFramerateLimit(60);
34      while (window.isOpen()) {
35          sf::Event event;
36          while (window.pollEvent(event)) {
37              if (event.type == sf::Event::Closed) {
38                  window.close();
39              }
40          }
41          window.clear(sf::Color::Blue);
42          for (int i = 0; i < triangleStorage.size();
                i++) {
43              window.draw(triangleStorage[i]);
44          }
45          window.display();
46      }
47      return 0;
48  }
49
50  void fTree(std::vector<Triangle> *triangleHouse, sf
```

```
    ::Vector2<double>
51 centerPoint, double base, int recur, sf::
    PrimitiveType type) {
52     if (recur < 0) {  // If number of recursions is
          negative, return
53         return;
54     }
55
56     if (base < 0) {  // If base is negative, set
          equal to 0
57         base = 0;
58     }
59     // Push current triangle into the vector for
          drawing
60     triangleHouse->push_back(Triangle(base,
          centerPoint, sf::LineStrip));
61
62     double Height = base * (sqrt(3) / 2);  //
          calculate Altitude
63
64     // Set VertexA
65     sf::Vector2<double> VertexA {(centerPoint.x -
          (0.5 * base)),
66     (centerPoint.y - (0.75 * Height))};
67     // Set VertexB
68     sf::Vector2<double> VertexB {(centerPoint.x +
          (0.75 * base)),
69     (centerPoint.y - (0.25 * Height))};
70     // Set VertexC
71     sf::Vector2<double> VertexC {(centerPoint.x -
          (0.25 * base)),
72     (centerPoint.y + (0.75 * Height))};
73
74     // recursive call for VertexA
75     fTree(triangleHouse, VertexA, (base / 2.0), (
          recur - 1), type);
76     // recursive call for VertexB
77     fTree(triangleHouse, VertexB, (base / 2.0), (
          recur - 1), type);
78     // recursive call for VertexC
```

```
79      fTree(triangleHouse, VertexC, (base / 2.0), (
            recur - 1), type);
80  }
```

4.5.3 Triangle.h

```
1  // (C) Copyright Erik van Tilborg, 2022.
2  // Distributed under MIT license, available at
3  // https://opensource.org/licenses/MIT.
4
5  #pragma once
6  #include <memory>
7  #include <SFML/Graphics.hpp>
8  #include <SFML/System.hpp>
9  #include <SFML/Window.hpp>
10
11 class Triangle: public sf::Drawable {
12  public:
13      Triangle(double base, sf::Vector2<double> center
            ,
14      sf::PrimitiveType = sf::LineStrip);
15
16  private:
17      std::shared_ptr<sf::VertexArray> verticies;
18
19      virtual void draw(sf::RenderTarget &target, sf::
            RenderStates states) const {  //NOLINT
20          target.draw(*verticies, states);
21      }
22 };
```

4.5.4 Triangle.cpp

```
1  // (C) Copyright Erik van Tilborg, 2022.
2  // Distributed under MIT license, available at
3  // https://opensource.org/licenses/MIT.
4
5  #include <iostream>
6  #include <cmath>
7  #include "Triangle.h"
8
```

```
 9 Triangle::Triangle(double base, sf::Vector2<double>
       cPoint,
10 sf::PrimitiveType type) {
11     double Alt = base * (sqrt(3) / 2);
12     verticies = std::make_shared<sf::VertexArray>(sf
           ::LinesStrip, 4);
13
14     (*verticies)[0].position =
15     sf::Vector2f(cPoint.x - (base / 2.0), cPoint.y -
           (Alt / 2.0));
16
17     (*verticies)[1].position =
18     sf::Vector2f(cPoint.x + (base / 2.0), cPoint.y -
           (Alt / 2.0));
19
20     (*verticies)[2].position =
21     sf::Vector2f(cPoint.x, cPoint.y + (Alt / 2.0));
22
23     (*verticies)[3] = (*verticies)[0];
24 }
```

5 PS4 - Synthesizing a Plucked String Sound

5.1 Overview

The assignment was to create a circular buffer that would be used in a final implementation of a program that emulates the plucking of a guitar string. It uses the circular buffer to make stringsound objects that house the tones for different notes based off of a premade array of characters that emulate a "keyboard".

5.2 Key Concepts

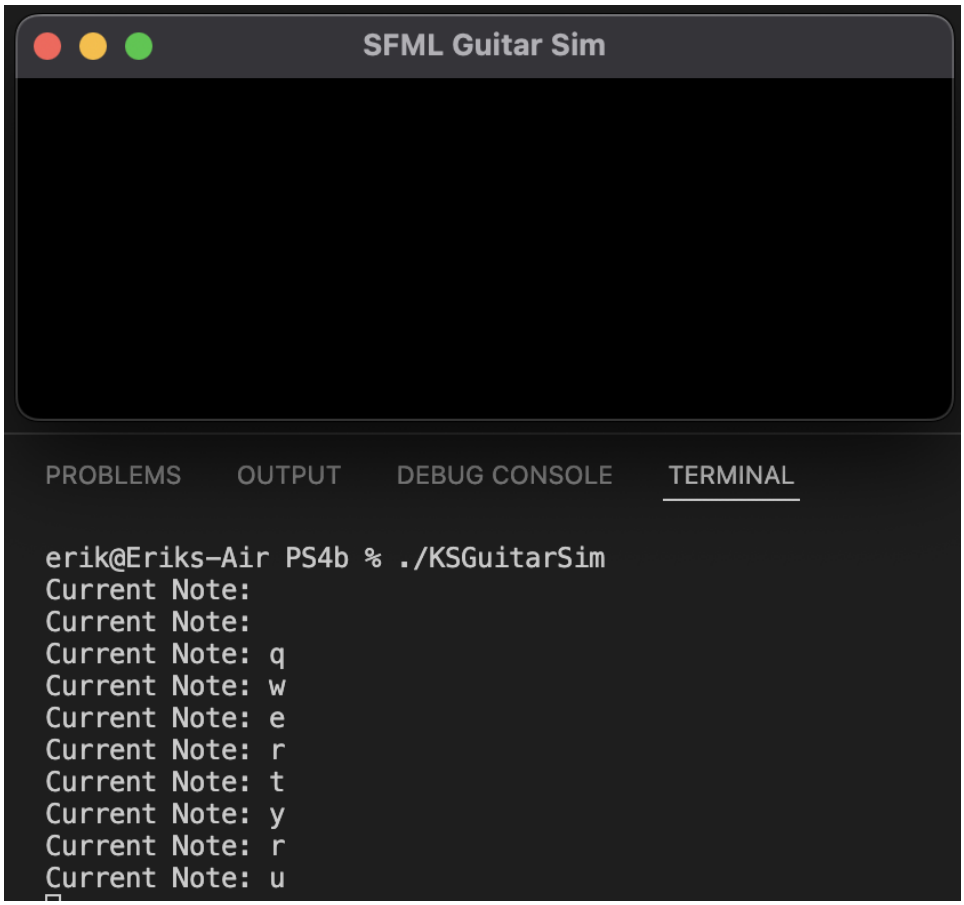
The key concepts for this assignment was implementing the two key classes that made the whole assignment. One for the CircularBuffer class and one for the StringSound class. The last one was implementing exceptions that would trigger if cases were not met or parameters were not getting expected values.

5.3 Learnings

In this assignment i learned how to make a circular buffer. then how to take that and implement it in a way that it pairs with a soundbuffer to create tones for notes based off of a starting frequency that can be played by pressing keys on the keyboard.

5.4 Output

Figure 5: SFML window running with terminal showing the noted played



5.5 Code

5.5.1 Makefile

```
CXX = g++
CXXFLAGS = -Wall -Werror -pedantic --std=c++17
LIBS = -lboost_unit_test_framework -lsfml-graphics -lsfml-audio -lsfml-system -lsfml-window
DEPS = CircularBuffer.h StringSound.h stringsoundtest.cpp circularbuffertest.cpp
LINTFILES = CircularBuffer.cpp CircularBuffer.h KSGuitarSim.cpp StringSound.cpp StringSound.h stringsoundtest.cpp

all:KSGuitarSim ssboosttest cbboosttest lint
%.o: %.cpp $(DEPS)
	$(CXX) $(CXXFLAGS) -I/opt/homebrew/Cellar/sfml/2.5.1_1/include -I/opt/homebrew/Cellar/boost/1.78.0_1/include -c $<

KSGuitarSim: KSGuitarSim.o CircularBuffer.o StringSound.o
	$(CXX) $(CXXFLAGS) -L/opt/homebrew/Cellar/sfml/2.5.1_1/lib -L/opt/homebrew/Cellar/boost/1.78.0_1/lib -o $@ $^ $(LIBS)

ssboosttest: stringsoundtest.o CircularBuffer.o StringSound.o
	$(CXX) $(CXXFLAGS) -L/opt/homebrew/Cellar/sfml/2.5.1_1/lib -L/opt/homebrew/Cellar/boost/1.78.0_1/lib -o $@ $^ $(LIBS)

cbboosttest: circularbuffertest.o CircularBuffer.o
	$(CXX) $(CXXFLAGS) -L/opt/homebrew/Cellar/sfml/2.5.1_1/lib -L/opt/homebrew/Cellar/boost/1.78.0_1/lib -o $@ $^ $(LIBS)

lint: $(LINTFILES)
	cpplint $(LINTFILES)

```

```
clean:
    rm -f *.o
    rm -f KSGuitarSim
    rm -f ssboosttest
```

5.5.2 KSGuitarSim.cpp

```
// (C) Copyright Erik van Tilborg, 2022
// Distributed under MIT license, available at
// https://opensource.org/licenses/MIT.

#include <string>
#include <SFML/Audio.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>

#include "StringSound.h"

std::vector<sf::Int16> makeSamples(StringSound*
    guitar) {
     std::vector<sf::Int16> samples;
     guitar->pluck();
     int duration = 8;
     int i;
     for (i= 0; i < SAMPLING_RATE * duration; i++) {
         guitar->tic();
         samples.push_back(guitar->sample());
     }
     return samples;
}

int main() {
  std::vector<std::vector<sf::Int16>> soundSamples;
  std::vector<sf::SoundBuffer> soundBuf(
      size_of_keyboard);
  std::vector<sf::Sound> sound(size_of_keyboard);

  soundSamples.reserve(size_of_keyboard);

  auto getRespectiveFrequency = [](auto i) {
```

```
32          return 440 * pow(2, ((i - 24) / 12));
33      };
34
35      for (int i = 0; i < size_of_keyboard; i++) {
36          StringSound freq(getRespectiveFrequency(i));
37          soundSamples[i] = makeSamples(&freq);
38          if (!soundBuf[i].loadFromSamples
39          (&soundSamples[i][0], soundSamples[i].size(),
               2, SAMPLING_RATE)) {
40              throw std::runtime_error("sf::SoundBuffer:
                    Failed to load");
41          }
42          sound[i].setBuffer(soundBuf[i]);
43      }
44
45      sf::RenderWindow window(sf::VideoMode(300, 300), "
           SFML Guitar Sim");
46      while (window.isOpen()) {
47        sf::Event event;
48        while (window.pollEvent(event)) {
49          if (event.type == sf::Event::Closed) {
50              window.close();
51          }
52          if (event.type == sf::Event::TextEntered) {
53            for (int i = 0; i < size_of_keyboard; i++) {
54              if (static_cast<unsigned>(event.text.
                    unicode) == keyboard_keys[i]) {
55                std::cout << "Current Note: " <<
                       keyboard_keys[i] << std::endl;
56                sound[i].play();
57              }
58            }
59          }
60          window.clear();
61          window.display();
62        }
63      }
64      return 0;
65  }
```

5.5.3 CircularBuffer.h

```
1 // (C) Copyright Erik van Tilborg, 2022
2 // Distributed under MIT license, available at
3 // https://opensource.org/licenses/MIT.
4
5 #pragma once
6
7 #include <stdint.h>
8 #include <iostream>
9 #include <memory>
10
11
12 class CircularBuffer {
13  public:
14   explicit CircularBuffer(int capacity);
15
16   ~CircularBuffer() = default;
17
18   size_t size() {
19     return capacity;
20   }
21
22   bool isEmpty();
23   bool isFull();
24   void enqueue(int16_t x);
25   int16_t dequeue();
26   int16_t peek();
27   void empty();
28
29   void setCapacity(int cap) {
30     if (cap <= 0) {
31       throw std::invalid_argument
32       ("Capacity must be greater than 0");
33     } else {
34       capacity = cap;
35     }
36   }
37
38   double getDecayVal() const {
```

```
39      return DECAY;
40    }
41
42   private:
43    std::unique_ptr<int16_t[]> ringBuffer;
44    size_t capacity;
45    size_t bufferFront;
46    size_t bufferBack;
47    size_t spacedUsed;
48    bool _full;
49
50    constexpr static double DECAY = 0.996;
51 };
```

5.5.4 CircularBuffer.cpp

```
1  // (C) Copyright Erik van Tilborg, 2022
2  // Distributed under MIT license, available at
3  // https://opensource.org/licenses/MIT.
4
5  #include "CircularBuffer.h"
6
7  CircularBuffer::CircularBuffer(int cap) {
8    capacity = cap;
9    bufferFront = 0;
10   bufferBack = 0;
11   spacedUsed = 0;
12   if (capacity < 1) {
13     throw std::invalid_argument
14     ("The capacity of the Buffer must be greater
          than 0!");
15   }
16   ringBuffer = std::make_unique<int16_t[]>(capacity)
       ;
17 }
18
19 bool CircularBuffer::isEmpty() {
20   if (spacedUsed != 0) {
21     _full = false;
22     return false;
```

```
23    } else {
24      _full = true;
25      return true;
26    }
27  }
28
29  bool CircularBuffer::isFull() {
30    if (spacedUsed == capacity) {
31      _full = true;
32      return true;
33    } else {
34      _full = false;
35      return false;
36    }
37  }
38
39  void CircularBuffer::enqueue(int16_t x) {
40    if (isFull()) {
41      throw std::runtime_error
42      ("The buffer is full... Cannot enqueue on a full
            buffer!");
43    }
44    ringBuffer[bufferBack] = x;
45    bufferBack = (bufferBack + 1) % capacity;
46    spacedUsed += 1;
47  }
48
49  int16_t CircularBuffer::dequeue() {
50    if (isEmpty()) {
51      throw std::runtime_error
52      ("The buffer is empty... Cannot dequeue from an
           empty buffer!");
53    }
54    int16_t front = ringBuffer[bufferFront];
55    bufferFront = (bufferFront + 1) % capacity;
56    spacedUsed -= 1;
57    return front;
58  }
59
60  int16_t CircularBuffer::peek() {
```

```
61   return ringBuffer[bufferFront];
62 }
63
64 void CircularBuffer::empty() {
65   bufferFront = 0;
66   bufferBack = 0;
67   spacedUsed = 0;
68   _full = false;
69 }
```

5.5.5 StringSound.h

```
1 // (C) Copyright Erik van Tilborg, 2022
2 // Distributed under MIT license, available at
3 // https://opensource.org/licenses/MIT.
4
5 #pragma once
6
7 #include "CircularBuffer.h"
8
9 #include <vector>
10 #include <cstdint>
11 #include <cmath>
12 #include <climits>
13 #include <iostream>
14 #include <ctime>
15 #include <random>
16
17 #include <SFML/Graphics.hpp>
18
19 static const auto CONCERT_A = 220.0;
20 static const auto SAMPLING_RATE = 44100;
21 static const auto size_of_keyboard = 37;
22 static const char keyboard_keys[] = "
     q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
23
24 class StringSound {
25  public:
26   explicit StringSound(double frequency);
27   explicit StringSound(std::vector<sf::Int16> init);
```

```
28
29   StringSound(const StringSound &obj) = delete;
30   ~StringSound();
31
32   void pluck();
33   void tic();
34
35   sf::Int16 sample();
36
37   int time();
38
39  private:
40   CircularBuffer * _cb;
41   int _time;
42 };
```

5.5.6 StringSound.cpp

```
1 // (C) Copyright Erik van Tilborg, 2022
2 // Distributed under MIT license, available at
3 // https://opensource.org/licenses/MIT.
4
5 #include "StringSound.h"
6
7 std::random_device device;
8 std::mt19937 generator(device());
9 std::uniform_int_distribution<int16_t> dist(SHRT_MIN
    , SHRT_MAX);
10
11 StringSound::StringSound(double frequency) {
12     if (frequency <= 0) {
13         throw std::invalid_argument
14         ("The frequency MUST be a number greater
             than zero");
15     } else {
16         auto capacity = std::ceil(SAMPLING_RATE /
             frequency);
17         _cb = new CircularBuffer(capacity);
18         _time = 0;
19     }
```

```
20 }
21
22 StringSound::StringSound(std::vector<sf::Int16> init
       ) {
23     if (init.size() <= 0) {
24         throw std::invalid_argument
25         ("The initial value given must be greater
              than 0");
26     } else {
27         _cb = new CircularBuffer(init.size());
28         _time = 0;
29         for (auto i : init) {
30             _cb->enqueue(i);
31         }
32     }
33 }
34
35 StringSound::~StringSound() {
36     delete _cb;
37 }
38
39 void StringSound::pluck() {
40     _cb->empty();
41     while (_cb->isFull() == false) {
42         _cb->enqueue(dist(generator));
43     }
44 }
45
46 void StringSound::tic() {
47     auto val1 = _cb->dequeue();
48     auto val2 = _cb->peek();
49     std::int16_t nextVal = _cb->getDecayVal() * 0.5
          * (val1 + val2);
50     _cb->enqueue(nextVal);
51     _time++;
52 }
53
54 sf::Int16 StringSound::sample() {
55     return _cb->peek();
56 }
```

```
57
58 int StringSound::time() {
59     return _time;
60 }
```

5.5.7 CircularBufferTest.cpp

```
1  // (C) Copyright Erik van Tilborg, 2022
2  // Distributed under MIT license, available at
3  // https://opensource.org/licenses/MIT.
4
5  #define BOOST_TEST_DYN_LINK
6  #define BOOST_TEST_MODULE test
7
8  #include "CircularBuffer.h"
9  #include <boost/test/unit_test.hpp>
10
11 BOOST_AUTO_TEST_CASE(enqueueAndDequeueTest) {
12     CircularBuffer buf(8);
13     BOOST_REQUIRE(buf.isEmpty());
14
15     buf.enqueue(1);
16     buf.enqueue(2);
17     buf.enqueue(3);
18     buf.enqueue(4);
19     buf.enqueue(5);
20     buf.enqueue(6);
21     buf.enqueue(7);
22     buf.enqueue(8);
23
24     BOOST_REQUIRE(buf.isFull());
25     BOOST_REQUIRE_EQUAL(buf.dequeue(), 1);
26     BOOST_REQUIRE_EQUAL(buf.dequeue(), 2);
27     BOOST_REQUIRE_EQUAL(buf.dequeue(), 3);
28     BOOST_REQUIRE_EQUAL(buf.dequeue(), 4);
29
30     buf.enqueue(9);
31     buf.enqueue(10);
32     buf.enqueue(11);
33
```

```
34      BOOST_REQUIRE(!buf.isEmpty());
35      BOOST_REQUIRE(!buf.isFull());
36  }
37
38  BOOST_AUTO_TEST_CASE(
        checkFrontAfterEnqueueAndDequeue) {
39      CircularBuffer buf(4);
40      buf.enqueue(4);
41      buf.enqueue(3);
42      buf.enqueue(2);
43      buf.enqueue(1);
44      BOOST_REQUIRE(buf.peek() == 4);
45
46      buf.dequeue();
47      buf.dequeue();
48      BOOST_REQUIRE(buf.peek() == 2);
49  }
50
51  BOOST_AUTO_TEST_CASE(testForOverflowError) {
52      CircularBuffer buf(6);
53      for (int i = 0; i < 5; i++) {
54          buf.enqueue(i);
55      }
56      BOOST_REQUIRE_NO_THROW(buf.enqueue(5));
57      BOOST_REQUIRE(buf.isFull());
58      BOOST_REQUIRE_THROW(buf.enqueue(6), std::
            runtime_error);
59  }
60
61  BOOST_AUTO_TEST_CASE(dequeueOnEmptyBuffer) {
62      CircularBuffer buf(10);
63      BOOST_REQUIRE_THROW(buf.dequeue(), std::
            runtime_error);
64  }
65
66  BOOST_AUTO_TEST_CASE(testForCapacityEqualTo0) {
67      BOOST_REQUIRE_THROW(CircularBuffer buf(0), std::
            invalid_argument);
68  }
```

5.5.8 StringSoundTest.cpp

```
1 // (C) Copyright Erik van Tilborg, 2022
2 // Distributed under MIT license, available at
3 // https://opensource.org/licenses/MIT.
4
5 #define BOOST_TEST_DYN_LINK
6 #define BOOST_TEST_MODULE test
7
8 #include "StringSound.h"
9 #include <boost/test/unit_test.hpp>
10
11 BOOST_AUTO_TEST_CASE(ExceptionHandling) {
12     // first tests if a negative value is passed in
        as an argument
13     BOOST_REQUIRE_THROW(StringSound t1(-1), std::
        invalid_argument);
14     // second tests the other constructor for an
        empty vector.
15     BOOST_REQUIRE_THROW(StringSound t2({}), std::
        invalid_argument);
16     /* given the nature of the random number
        generator that is used
17     in the StringSound.cpp file and used in the
        pluck function, i
18     could not think of clear test that would produce
         the same result
19     everytime it would be tested. */
20 }
```

6 PS5 - DNA Sequence Alignment

6.1 Overview

The goal of this assignment was to create a matrix based on two DNA sequences and see where they would line up in the sequence. Calculations had to be made if they sequences mismatched or had a gap from one to the other and would be reflected in the final edit distance.

6.2 Key Concepts

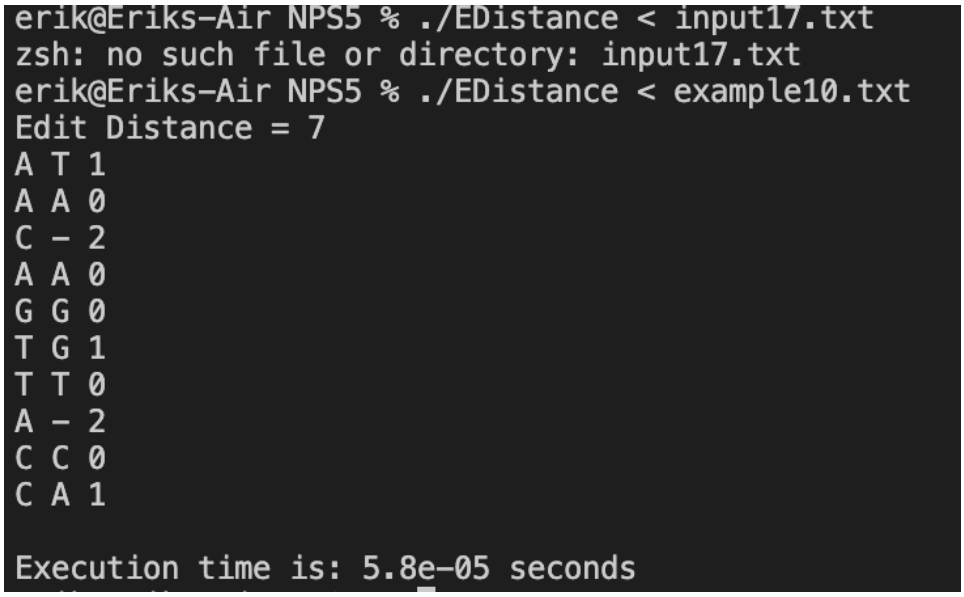
The key concepts for this assignment was creating the matrix needed to do the calculations. I used a vector of vectors to emulate what a matrix is in memory

6.3 Learnings

The main takeaways from this assignment was getting the matrix implementation correct to result int the correct edit distance. It was also to get the correct sequence alignment between the two DNA sequences.

6.4 Output

Figure 6: DNA sequencing results of example10.txt



6.5 Code

6.5.1 Makefile

```
CXX = g++
CXXFLAGS = -Wall -Werror -pedantic --std=c++17
LIBS = -lsfml-system
DEPS = EDistance.h
LINTFILES = EDistance.h EDistance.cpp main.cpp

all:EDistance lint
%.o: %.cpp $(DEPS)
    $(CXX) $(CXXFLAGS) -I/opt/homebrew/Cellar/sfml
       /2.5.1_1/include -c $<

EDistance: main.o EDistance.o
    $(CXX) $(CXXFLAGS) -L/opt/homebrew/Cellar/sfml
       /2.5.1_1/lib -o $@ $^ $(LIBS)

lint: $(LINTFILES)
    cpplint $(LINTFILES)

clean:
    rm -f *.o
    rm -f EDistance
```

6.5.2 main.cpp

```
// (C) Copyright Erik van Tilborg, 2022.
// Distributed under MIT license, available at
// https://opensource.org/licenses/MIT.

#include "EDistance.h"
#include <SFML/System.hpp>

int main() {
    sf::Clock clock;
    sf::Time t;

    std::string str1, str2;
    std::cin >> str1;
```

```
14      std::cin >> str2;
15
16      EDistance ed(str1, str2);
17
18      int optDist = ed.optDistance();
19      std::string Alignment = ed.alignment();
20
21      t = clock.getElapsedTime();
22
23      std::cout << "Edit Distance = "  << optDist <<
            std::endl;
24      std::cout << Alignment << std::endl;
25      std::cout << "Execution time is: " << t.
            asSeconds() << " seconds \n";
26
27      return 0;
28  }
```

6.5.3 EDistance.h

```
1  // (C) Copyright Erik van Tilborg, 2022.
2  // Distributed under MIT license, available at
3  // https://opensource.org/licenses/MIT.
4  #pragma once
5
6  #include <iostream>
7  #include <vector>
8  #include <string>
9
10 class EDistance {
11  public:
12   EDistance(std::string str_a, std::string str_b);
13   static int penalty(char a, char b);
14   static int min(int a, int b, int c);  //NOLINT
15   int optDistance();
16   std::string alignment();
17   void printMatrix();
18
19  private:
20   std::string A_str;
```

```
21   std::string B_str;
22   std::vector< std::vector<int> > mtx;
23 };
```

6.5.4 EDistance.cpp

```
1  // (C) Copyright Erik van Tilborg, 2022.
2  // Distributed under MIT license, available at
3  // https://opensource.org/licenses/MIT.
4
5  #include "EDistance.h"
6  #include <string.h>
7  #include <iomanip>  // used in conjunction with
       debugging matrix output
8
9  const int MATCH = 0;
10 const int MISMATCH = 1;
11 const int GAP = 2;
12
13 EDistance::EDistance(const std::string strA, const
       std::string strB) :
14                       A_str(strA), B_str(strB), mtx(
                             A_str.size() + 1,
15                       std::vector<int>(B_str.size() +
                             1, 0)) {
16     int emuMult2 = 0;
17     for (int i = strB.size(); i >= 0 ; i--) {
18         mtx[strA.size()][i] = emuMult2;
19         emuMult2 += 2;
20     }
21
22     emuMult2 = 0;
23     for (int j = strA.size(); j >= 0; j--) {
24         mtx[j][strB.size()] = emuMult2;
25         emuMult2 += 2;
26     }
27 }
28
29 int EDistance::penalty(char a, char b) {
30     return (a == b) ? MATCH : MISMATCH;  // ternary
```

```
        op condenses if-else block
}

int EDistance::min(int a, int b, int c) {  //NOLINT
    if (a <= b && a <= c) {             // check one
        condition
        return a;
    } else {  // use of ternary op cuts down on if-
        else counts to check final 2
        return (b <= c) ? b : c;
    }
}

int EDistance::optDistance() {
    for (int i = A_str.size() - 1; i >= 0; i--) {
        for (int j = B_str.size() - 1; j >= 0; j--)
            {
            mtx[i][j] = min(mtx[i + 1][ j+ 1] +
                penalty(A_str[i], B_str[j]),  //
                NOLINT
            mtx[i + 1][j] + GAP, mtx[i][j + 1] + GAP
                );  //NOLINT
        }
    }
    return mtx[0][0];
}

std::string EDistance::alignment() {
    unsigned int i = 0, j = 0;
    std::string retString;
    while (i < A_str.size() && j < B_str.size()) {
        if (mtx[i][j] == mtx[i + 1][j + 1] + penalty
            (A_str[i], B_str[j])) {
            retString.push_back(A_str[i]);
            retString.push_back(' ');
            retString.push_back(B_str[j]);
            retString.push_back(' ');
            retString.push_back((penalty(A_str[i],
                B_str[j])) + '0');
            retString.push_back('\n');
```

```
62              i++;
63              j++;
64          } else if (mtx[i][j] == mtx[i + 1][j] + GAP)
                {
65              retString.push_back(A_str[i]);
66              retString.push_back(' ');
67              retString.push_back('-');
68              retString.push_back(' ');
69              retString.push_back(GAP + '0');
70              retString.push_back('\n');
71              i++;
72          } else if (mtx[i][j] == mtx[i][j + 1] + GAP)
                {
73              retString.push_back('-');
74              retString.push_back(' ');
75              retString.push_back(B_str[j]);
76              retString.push_back(' ');
77              retString.push_back(GAP + '0');
78              retString.push_back('\n');
79              j++;
80          }
81      }
82      return retString;
83  }
84
85  // used for debugging the matrix values
86  void EDistance::printMatrix() {
87      for (unsigned i = 0; i < mtx.size(); i++) {
88          for (unsigned j = 0; j < mtx[i].size(); j++)
                {
89              std::cout << mtx[i][j] << std::setw(4);
90          }
91          std::cout << std::endl;
92      }
93  }
```

7 PS6 - Random Writer

7.1 Overview

The goal of this assignment was to use the Markov Model to generate kgrams of a given order, keep track of their frequency as well as the frequency of the character that comes after to generate a random output based on an input given.

7.2 Key Concepts

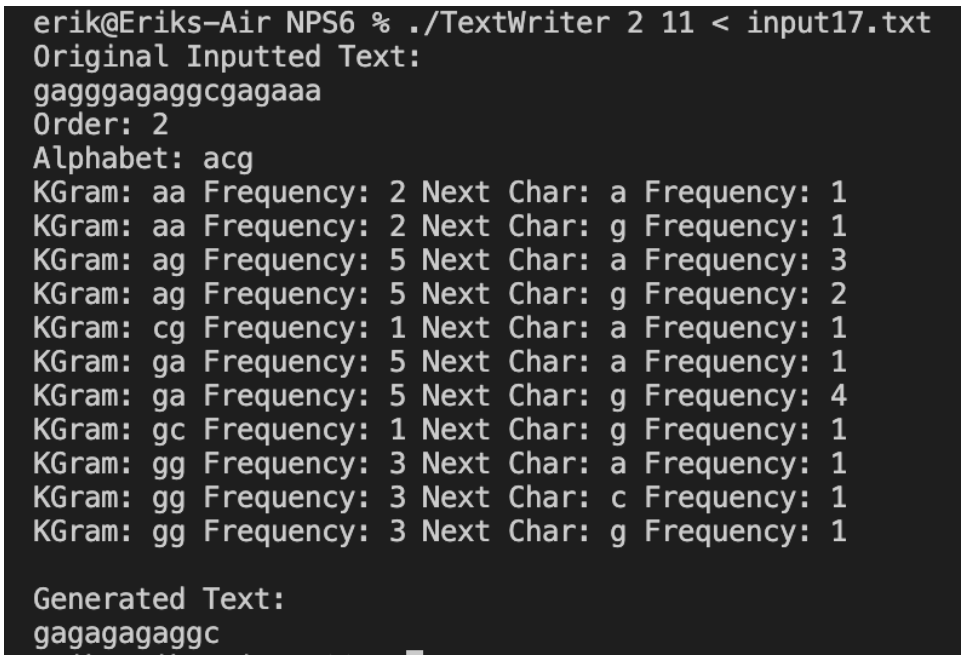
The key concept of this assignment was to get the kgram generator working for use in the final implementation. The kgram generator was then used to make kgrams according to a user specified order to make pseudo random text from an input fed in by the command line.

7.3 Learnings

The main takeaways from this assignment was creating the kgram generator and getting that to work. The assignment also gave an idea of how this could be used to encode text with the pseudo random text that is generated as a result of the kgrams.

7.4 Output

Figure 7: Image showing the kgrams generated, frequencies and the pseudo random text generated



7.5 Code

7.5.1 Makefile

```
CXX = g++
CXXFLAGS = -g -Wall -Werror -pedantic --std=c++17
LIBS = -lboost_unit_test_framework
DEPS = RandWriter.h
LINTFILES = RandWriter.h RandWriter.cpp TextWriter.
    cpp test.cpp

all:TextWriter BoostTest lint
%.o: %.cpp $(DEPS)
    $(CXX) $(CXXFLAGS) -I/opt/homebrew/Cellar/boost
       /1.78.0_1/include -c $<

TextWriter: TextWriter.o RandWriter.o
    $(CXX) $(CXXFLAGS) -o $@ $^

BoostTest: test.o RandWriter.o
    $(CXX) $(CXXFLAGS) -L/opt/homebrew/Cellar/boost
       /1.78.0_1/lib -o $@ $^ $(LIBS)

lint: $(LINTFILES)
    cpplint $(LINTFILES)

clean:
    rm -f *.o
    rm -f TextWriter
    rm -f BoostTest
```

7.5.2 TextWriter.cpp

```
// (C) Copyright Erik van Tilborg, 2022.
// Distributed under MIT license, available at
// https://opensource.org/licenses/MIT.

#include <iostream>
#include <string>
#include "RandWriter.h"

```

```
 9 using std::cout;
10
11 template <typename T>
12 int convertCharToInt(const char * argument, T&&
      rickroll) {
13     auto temp = rickroll;
14     return temp(argument);
15 }
16
17 int main(int argc, const char * argv[]) {
18     if (argc != 3) {
19         cout << "Usage is as follows: ./
              TextGenerator <int k> <int L>\n";
20         return -1;
21     }
22
23     int k = convertCharToInt(argv[1], [&](auto temp)
          { return std::stoi(temp);});
24     int L = convertCharToInt(argv[2], [&](auto temp)
          {return std::stoi(temp);});
25
26
27     std::string text;
28     std::string input_text = "";
29     std::string output_text = "";
30
31     while (std::getline(std::cin, input_text)) {
32         text.append(input_text);
33         input_text = "";
34     }
35
36     cout << "Original Inputted Text:\n";
37     for (int i = 0; i < text.length(); i++) {
38         cout << text[i];
39     }
40     cout << std::endl;
41
42     RandWriter rw(text, k);
43
44     cout << rw << std::endl;
```

```
45
46      output_text = rw.generate(text.substr(0, k), L);
47
48      cout << "Generated Text:\n";
49      for (int i = 0; i < L; i++) {
50          cout << output_text[i];
51      }
52      cout << std::endl;
53
54      // debugOutput(rw.getMap());
55
56      return 0;
57  }
```

7.5.3 RandWriter.h

```
1  // (C) Copyright Erik van Tilborg, 2022.
2  // Distributed under MIT license, available at
3  // https://opensource.org/licenses/MIT.
4
5  #pragma once
6
7  #include <string>
8  #include <map>
9  #include <algorithm>
10 #include <stdexcept>
11 #include <utility>
12 #include <vector>
13 #include <iostream>
14 #include <functional>
15
16 struct kGramInfo {
17     int freq;
18     std::map <char, int> charFreq;
19 };
20
21 class RandWriter {
22  public:
23   RandWriter(std::string text, int k);
24   int orderK() const;
```

```
25   int freq(std::string kgram) const;
26   int freq(std::string kgram, char c) const;
27   char kRand(std::string kgram);
28   std::string generate(std::string kgram, int L);
29   friend std::ostream& operator<<(std::ostream &out,
         RandWriter &obj);
30  private:
31   int _order;
32   std::string Alphabet;
33   std::map <std::string, kGramInfo> kGramMap;
34 };
```

7.5.4 RandWriter.cpp

```
1  // (C) Copyright Erik van Tilborg, 2022.
2  // Distributed under MIT license, available at
3  // https://opensource.org/licenses/MIT.
4
5  #include "RandWriter.h"
6  #include <iterator>
7  #include <vector>
8
9  RandWriter::RandWriter(std::string text, int k) :
       _order(k) {
10     int lengthOfText = text.size();
11     char temp;
12     bool alpha = false;
13     std::string copyText = text;
14
15     for (int i = 0; i < _order; i++) {
16         copyText.push_back(text[i]);
17     }
18
19     for (int i = 0; i < lengthOfText; i++) {
20         temp = text.at(i);
21         alpha = false;
22         for (unsigned int j = 0; j < Alphabet.length
               (); j++) {
23             if (Alphabet.at(j) == temp) {
24                 alpha = true;
```

```
25              }
26          }
27          if (!alpha) {
28              Alphabet.push_back(temp);
29          }
30      }
31
32      std::sort(Alphabet.begin(), Alphabet.end());
33      std::string subStr;
34      char nextChar;
35
36      for (int j = 0; j < text.length(); j++) {
37          nextChar = copyText.at(j + _order);
38          subStr = copyText.substr(j, _order);
39          kGramMap[subStr].freq++;
40          kGramMap[subStr].charFreq[nextChar];
41          kGramMap[subStr].charFreq[nextChar]++;
42      }
43
44      srand((int)time(NULL));  //NOLINT
45  }
46
47  int RandWriter::orderK() const {
48      return _order;
49  }
50
51  int RandWriter::freq(std::string kgram) const {
52      if (kgram.length() != _order) {
53          throw std::runtime_error("ERROR: kGram not
                of length k");
54      }
55
56      std::map<std::string, kGramInfo>::const_iterator
            iter;
57      iter = kGramMap.find(kgram);
58      if (iter == kGramMap.end()) {
59          return 0;
60      }
61      return kGramMap.at(kgram).freq;
62  }
```

```
63
64 int RandWriter::freq(std::string kgram, char c)
       const {
65     if (kgram.length() != _order) {
66         throw std::runtime_error("ERROR: kGram not
               of length k");
67     }
68
69     std::map<std::string, kGramInfo>::const_iterator
           iter;
70     iter = kGramMap.find(kgram);
71     if (iter == kGramMap.end()) {
72         return 0;
73     }
74     return kGramMap.at(kgram).charFreq.at(c);
75 }
76
77 char RandWriter::kRand(std::string kgram) {
78     if (kgram.length() != _order) {
79         throw std::runtime_error("ERROR: kgram is
               not of length k");
80     }
81
82     std::map<std::string, kGramInfo>::iterator iter;
83     std::map<char, int>::iterator iter2;
84
85     iter = kGramMap.find(kgram);
86
87     if (iter == kGramMap.end()) {
88         throw std::runtime_error("Inputted kGram
               does not exist");
89     }
90
91     std::vector<char> randData;
92
93     for (iter2 = iter->second.charFreq.begin();
94          iter2 != iter->second.charFreq.end(); iter2
                 ++) {
95         for (int i = 0; i < iter2->second; i++) {
96             randData.push_back(iter2->first);
```

```
97          }
98      }
99
100     int randVal = (rand() % randData.size());  //
            NOLINT
101
102     return randData[randVal];
103 }
104
105 std::string RandWriter::generate(std::string kgram,
        int L) {
106     if (kgram.length() != _order) {
107         throw std::runtime_error("ERROR: kgram is
                not of length k");
108     }
109     if (L <= _order) {
110         throw std::runtime_error("ERROR: L must be
                greater >= to k");
111     }
112     std::string returnStr = kgram;
113     char returnChar;
114
115     for (int i = 0; i < (L - _order); i++) {
116         returnChar = kRand(returnStr.substr(i,
                _order));
117         returnStr.push_back(returnChar);
118     }
119
120     return returnStr;
121 }
122
123 std::ostream &operator<<(std::ostream &out,
        RandWriter &obj) {
124     out << "Order: " << obj.orderK() << std::endl;
125     out << "Alphabet: " << obj.Alphabet << std::endl
            ;
126     std::map<std::string, kGramInfo>::iterator KIter
            ;
127     std::map<char, int>::iterator CFIter;
128     for (KIter = obj.kGramMap.begin(); KIter != obj.
```

```
        kGramMap.end(); KIter++) {
129         for (CFIter = KIter->second.charFreq.begin()
                ;
130              CFIter != KIter->second.charFreq.end();
                     CFIter++) {
131             out << "KGram: " << KIter->first << "
                    Frequency: "
132             << KIter->second.freq << " Next Char: "
                    << CFIter->first
133             << " Frequency: " << CFIter->second <<
                    std::endl;
134         }
135     }
136     return out;
137 }
```

7.5.5 Test.cpp

```
 1 // (C) Copyright Erik van Tilborg, 2022.
 2 // Distributed under MIT license, available at
 3 // https://opensource.org/licenses/MIT.
 4
 5 #define BOOST_TEST_DYN_LINK
 6 #define BOOST_TEST_MODULE unit_testing
 7 #include <boost/test/unit_test.hpp>
 8 #include "RandWriter.h"
 9
10 BOOST_AUTO_TEST_CASE(order2TestBasedOnStringFromPDF)
       {
11     BOOST_REQUIRE_NO_THROW(RandWriter("
          gagggagaggcgagaaa", 2));
12
13     RandWriter rw("gagggagaggcgagaaa", 2);
14
15     BOOST_REQUIRE(rw.orderK() == 2);
16
17     // throws error, kgram doesnt exist
18     BOOST_REQUIRE_THROW(rw.freq(""), std::
          runtime_error);
19     BOOST_REQUIRE_THROW(rw.freq("a"), std::
```

```
        runtime_error);
20
21      // throws error, kgram is not of length k
22      BOOST_REQUIRE_THROW(rw.freq("", 'g'), std::
            runtime_error);
23      BOOST_REQUIRE_THROW(rw.freq("a", 'g'), std::
            runtime_error);
24      BOOST_REQUIRE_THROW(rw.freq("aaa", 'g'), std::
            runtime_error);
25
26      // no throw, kgram exists in the map
27      BOOST_REQUIRE_NO_THROW(rw.freq("aa"));
28
29      // aa kGram tests
30      BOOST_REQUIRE(rw.freq("aa") == 2);
31      BOOST_REQUIRE(rw.freq("aa", 'a') == 1);
32      BOOST_REQUIRE(rw.freq("aa", 'g') == 1);
33
34      // ag kGram tests
35      BOOST_REQUIRE(rw.freq("ag") == 5);
36      BOOST_REQUIRE(rw.freq("ag", 'a') == 3);
37      BOOST_REQUIRE(rw.freq("ag", 'g') == 2);
38
39      // cg kGram tests
40      BOOST_REQUIRE(rw.freq("cg") == 1);
41      BOOST_REQUIRE(rw.freq("cg", 'a') == 1);
42
43      // ga kGram tests
44      BOOST_REQUIRE(rw.freq("ga") == 5);
45      BOOST_REQUIRE(rw.freq("ga", 'a') == 1);
46      BOOST_REQUIRE(rw.freq("ga", 'g') == 4);
47
48      // gc kGram tests
49      BOOST_REQUIRE(rw.freq("gc") == 1);
50      BOOST_REQUIRE(rw.freq("gc", 'g') == 1);
51
52      // gg kGram tests
53      BOOST_REQUIRE(rw.freq("gg") == 3);
54      BOOST_REQUIRE(rw.freq("gg", 'a') == 1);
55      BOOST_REQUIRE(rw.freq("gg", 'c') == 1);
```

```
56      BOOST_REQUIRE(rw.freq("gg", 'g') == 1);
57
58      // kRand tests
59      BOOST_REQUIRE_NO_THROW(rw.kRand("aa"));
60
61      BOOST_REQUIRE_THROW(rw.kRand("aaa"), std::
            runtime_error);
62
63      // supplied kgram is not of length k
64      BOOST_REQUIRE_THROW(rw.kRand("aaa"), std::
            runtime_error);
65
66      BOOST_REQUIRE_NO_THROW(rw.generate("aa", 11));
67
68      BOOST_REQUIRE_THROW(rw.generate("aa", 1), std::
            runtime_error);
69  }
```

8 PS7 - Kronos Time Clock

8.1 Overview

The assignment was and introduction to the Kronos time clock and was to make a program that finds when a server boots on and completes its boot sequence, calculate the boot time in milliseconds

8.2 Key Concepts

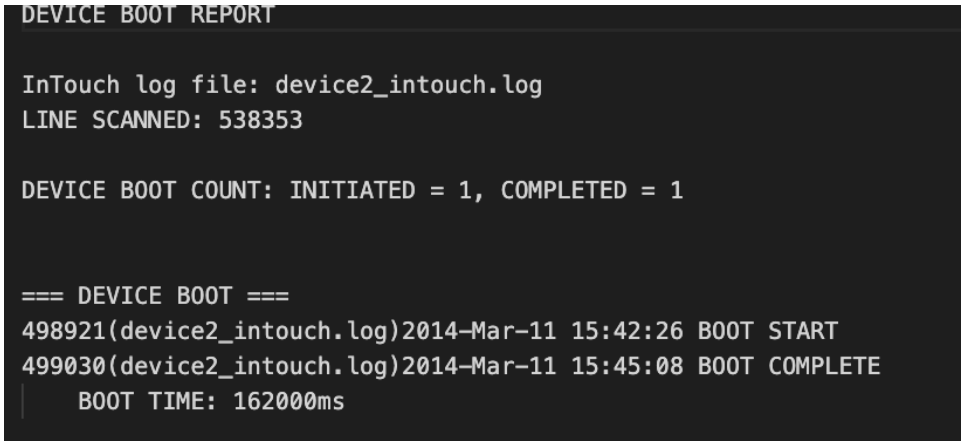
The key concepts for this was the date/time and regex part of the BOOST libraries to get the timestamps and seeing when the boot sequence starts based on a constant string.

8.3 Learnings

What I learned from this assignment was how to use the boost::regex functions as well as some functions from the date/time portion of the BOOST libraries.

8.4 Output

Figure 8: Result from the device2 intouch log file



8.5 Code

8.5.1 Makefile

```
CXX = g++
CXXFLAGS = -g -Wall -Werror -pedantic --std=c++17
LIBS = -lboost_regex -lboost_date_time
LINTFILES = main.cpp

all:ps7 lint
%.o: %.cpp
    $(CXX) $(CXXFLAGS) -I/opt/homebrew/Cellar/boost
       /1.78.0_1/include -c $<

ps7: main.o
    $(CXX) $(CXXFLAGS) -L/opt/homebrew/Cellar/boost
       /1.78.0_1/lib -o $@ $^ $(LIBS)

lint: $(LINTFILES)
    cpplint $(LINTFILES)

clean:
    rm -f *.o
    rm -f ps7
```

8.5.2 main.cpp

```
// (C) Copyright Erik van Tilborg, 2022.
// Distributed under MIT license, available at
// https://opensource.org/licenses/MIT.

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>

#include <boost/regex.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

int main(int argc, const char* argv[]) {
```

```
14      // const values for start of server and end of
           server within the log files
15      // "\\" included in the string b/c it couldnt
           find it as (log.c.166).
16      const boost::regex bootStart("\\(log\\.c
           \\.166\\) server started");
17      const boost::regex bootEnd
18                      ("oejs\\.AbstractConnector:
                           Started SelectChannelConnector
                           ");
19
20      // if only the executable is put in the command
           line
21      if (argc <= 1) {
22          std::cout << "ERROR: No file was passed into
                the program" << std::endl;
23          return -1;
24      }
25
26      // if more than the executable and a file are
           passed in
27      if (argc > 2) {
28          std::cout << "ERROR: Too many arguments in
                command line" << std::endl;
29          return -1;
30      }
31
32      std::string filePath(argv[1]);  // string for
           name of filePath
33      std::ifstream logPath(filePath);  // file in
           from the filepath
34      std::ofstream outputFile(filePath + ".rpt");  //
            output file
35      std::stringstream rpt;
36
37      // check to see the file status
38      if (logPath.is_open() == false) {
39          std::cout << "ERROR: The requested file
                cannot be opened: "
40                    << filePath << std::endl;
```

```
41          return -1;
42      }
43
44      bool bootSeq = false;
45      std::string currline;
46      int lineNum = 1;
47      int bootSuccessCount = 0;
48      int bootFailCount = 0;
49
50      /*
           ************************************************************

51      convienent lambda function that will be usefull
           later to get the
52      timestamps of the date and time accessed in
           kronos since they
53      show up in the same position, i.e. at the
           beginning of a line
54      ************************************************************
           */
55      auto extractTimeStamp = [](std::string str) {
56          return str.substr(0, 19);
57      };
58
59      boost::posix_time::ptime start, end;
60
61      while (std::getline(logPath, currline)) {  //
           while getline...
62          // if it finds the conditions for a boot
              start in the current line...
63          if (boost::regex_search(currline, bootStart)
              ) {
64              start = boost::posix_time::
                 time_from_string
65                      (extractTimeStamp(currline));
                           // get timestamp from lambda
66              // if the system is in a boot sequence,
                 result is an incomplete boot
67              if (bootSeq == true) {
68                  rpt << "**** INCOMPLETE BOOT ****\n\
```

```
                    n";
69                  ++bootFailCount;  // inc bootFail
70              }
71              bootSeq = true;  // sets boot sequence
                    flag to in boot
72              // otherwise: record boot sequence in
                    the log report
73              rpt << "=== DEVICE BOOT ===\n"
74                  << lineNum << "(" << filePath << ")"
75                  << start << " BOOT START\n";
76          }
77
78          // if it finds the conditions for a boot end
                in the current line...
79          if (boost::regex_search(currline, bootEnd))
                {
80              end = boost::posix_time::
                    time_from_string
81                      (extractTimeStamp(currline));  //
                            get timestamp from lambda
82              bootSeq = false;  // sets bootSeq flag
                    to not in boot
83              // gets total elapsed time by taking end
                    - start
84              boost::posix_time::time_duration
                    elapsedTime = end - start;
85              ++bootSuccessCount;  // inc bootSuccess
86              // record everything in the report log
87              rpt << lineNum << "(" << filePath << ")"
88                  << end << " BOOT COMPLETE\n\tBOOT
                        TIME: "
89                  << elapsedTime.total_milliseconds()
                        << "ms\n\n";
90          }
91          lineNum++;  // inc lineNum
92      }
93
94      // variable for total amount of boot sequences
95      int totalBoots = (bootSuccessCount +
            bootFailCount);
```

```
96
97      // finally record everything to the output file
            along with the report data
98      outputFile << "DEVICE BOOT REPORT\n\n" << "
            InTouch log file: " << filePath
99                 << "\n" << "LINE SCANNED: " <<
                       lineNum << "\n\n"
100                << "DEVICE BOOT COUNT: INITIATED = "
                       << totalBoots
101                << ", COMPLETED = " <<
                       bootSuccessCount << "\n\n\n"
102                << rpt.rdbuf();
103
104     // closing the files
105     outputFile.close();
106     logPath.close();
107
108     return 0;
109 }
```