Jacob Leboeuf

COMP IV: Project Portfolio

Fall 2020

Contents:

PS0: Hello World with SFML

Assignment Description

This assignment was an overall introduction to the course, and a way to get us used to the basic features of the Simple Fast Multimedia Library, or SFML. Such features include allowing the user to interact with the keyboard, import image media, and drawing and manipulating objects in the display area.

In my implementation for this assignment, I loaded a random image I chose named “sprite.png” into the program itself and allowed it to respond to various keystrokes. The image would move up, left, down, and right corresponding to pressing the W, A, S, and D keys, and would rotate the image with every keystroke pressed.

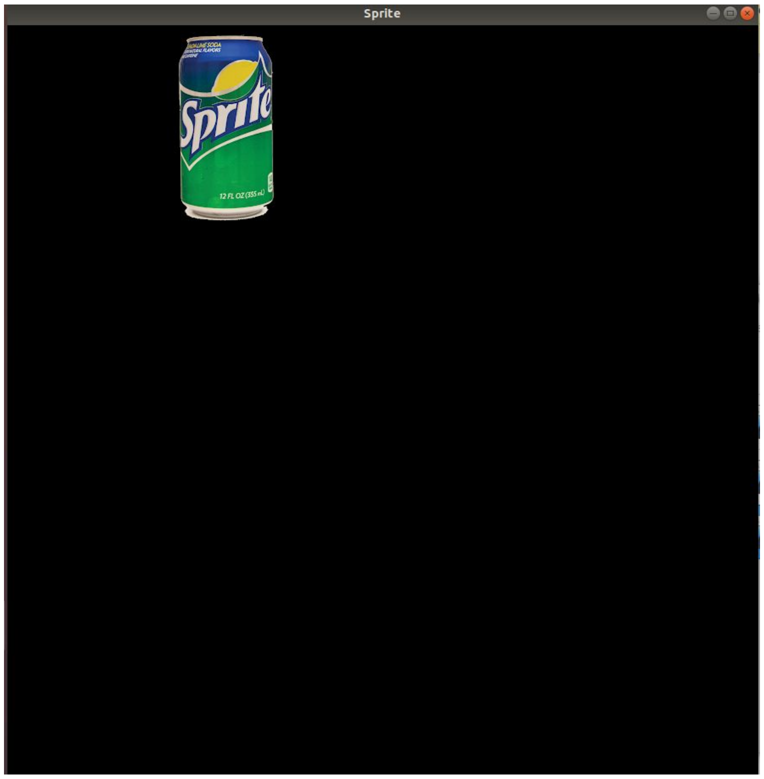
Key Algorithms, Data Structures, OO Design, Etc.

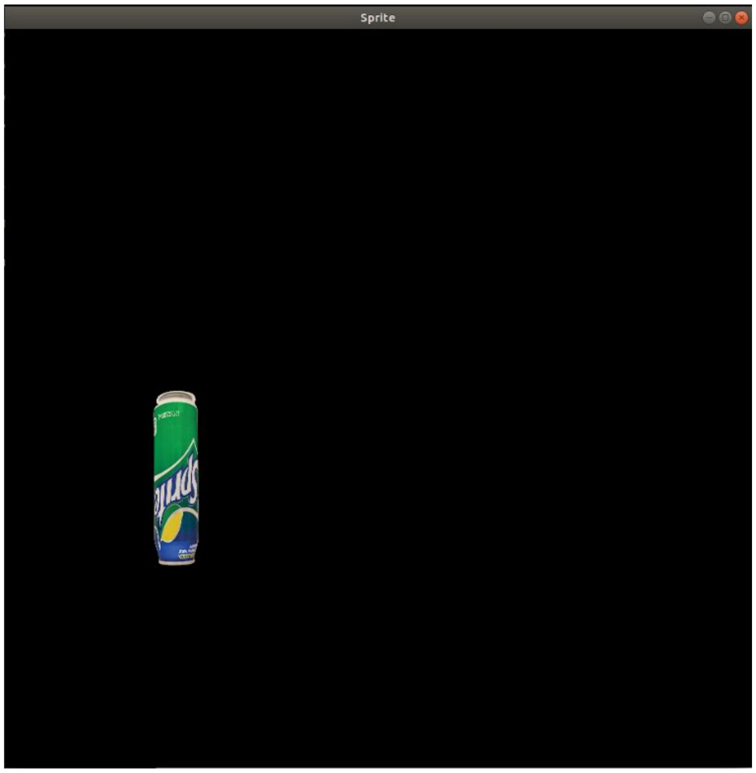
A key aspect this assignment had us work with and take use of was the usage of SFML’s window object as well as its render loop. This loop would run as long as the window is open, and would check to see what events were caught by the built in event handler with every frame. This event handler returns an Event object whose type we can read and use to determine if and how to update the sprite being drawn to the output screen. After this check is finished, the updated sprite is redrawn, and the process is repeated.

What I’ve Learned

As I was unfamiliar with the SFML library entirely prior to taking this course, this assignment did a very good job helping me learn the basics of the library. The new knowledge learned and experience gained with the window’s render loop would be helpful to me in several proceeding assignments that involved the usage of the library itself, either including using keyboard inputs or displaying objects, or just as a simple structure to use with projects needing different render loops.

Screenshot(s) of Output:





Source Code:

Makefile:

```
sfml-app: main.o
	g++ main.o -o sfml-app -lsfml-graphics -lsfml-window -lsfml-system

main.o: main.cpp
	g++ -c -Wall -Werror -ansi -pedantic main.cpp

clean:
	rm main.o main.cpp
```

main.cpp:

```
// Jacob Leboeuf
// 9-9-20
// Comp IV
#include <SFML/Graphics.hpp>
#include <iostream>
using namespace sf;
int main()
{
    // Create the main window
    sf::RenderWindow window(sf::VideoMode(1500, 1000), "Sprite");
    window.setFramerateLimit(60);
    // Load a sprite to display
    sf::Texture texture;
    if (!texture.loadFromFile("sprite.png"))
        return EXIT_FAILURE;
    sf::Sprite sprite(texture);
    // Start the game loop
    while (window.isOpen())
    {
        // Process events
        sf::Event event;
        while (window.pollEvent(event))
        {
            // Close window: exit
            if (event.type == Event::Closed)
                window.close();
            //move the image
            sprite.move(1,1);
            // move the image via keyboard
            if (event.type == Event::KeyPressed) {
                 switch (event.key.code) {
                        case Keyboard::W: sprite.move(0,-5); break;
                        case Keyboard::A: sprite.move(-5,0); break;
                        case Keyboard::S: sprite.move(0,5); break;
                        case Keyboard::D: sprite.move(5,0); break;
                        default: break;
                 }
            }
            // (something else) rotate the image
            sprite.rotate(90);
        }
        // Clear screen
        window.clear();
        // Draw the sprite
        window.draw(sprite);
        // Update the window
        window.display();
```

PS1: Linear Feedback Shift Register

Assignment Description

This assignment involves implementing a Linear Feedback Shift Register class to generate pseudo-random bits, and providing a basic form of encryption for digital images by using that register. It also involved using the Boost test framework to ensure that the created LFSR class would operate properly and produce the correct desired output.
The LFSR itself takes a linear function of a previous state as an input, performing discrete step operations that shift the bits one position to the left, and replaces the vacated bit by the XOR of the bit shifted off and the bit previously at a given tap position in the register.

In order to use this to encrypt an image, we must start by initializing the register with a set state, which will act as the encryption key. Due to each pixel within an image being represented by 8-bit values, we can use the LFSR to repeatedly generate 8-bit integers and overwrite the original pixel values with the bitwise XOR of it and the generated integer value. As a result, the final product will be an image that is garbled and encrypted, and the process to ungarble and decrypt the original image is by simply running the process again with the encrypted image.

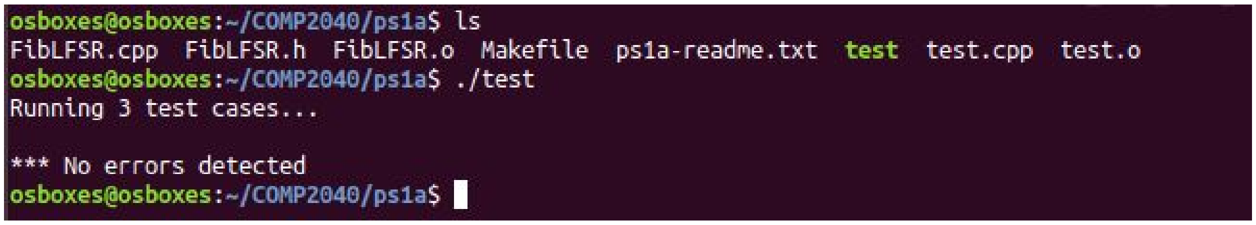
Key Algorithms, Data Structures, OO Design, Etc.

The data structure used within the assignment was a vector of booleans, where each boolean within the vector represented a bit in the LFSR with a true/false, or 1/0 value. This was used to hold the values of all the bits in the register, and allowed me to use various vector operations like pop_back and insert() in order to remove bits from the back of the register and insert them in the front of it.

What I’ve Learned

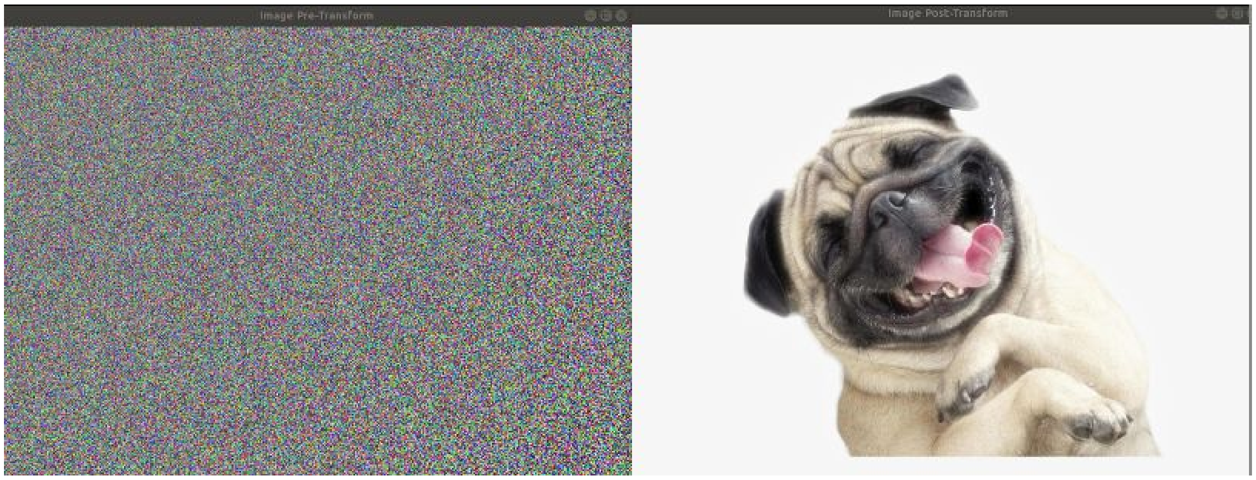
This assignment helped me gain more experience and general knowledge with bitwise operators and taught me the basics of the Boost unit testing framework. Considering that this project required much bitwise shifting and the usage of the XOR function, this overall helped me become more efficient with dealing with binary values when coding. Lastly, PS1 had me gain more experience with using the command line to read arguments for input as well as output.

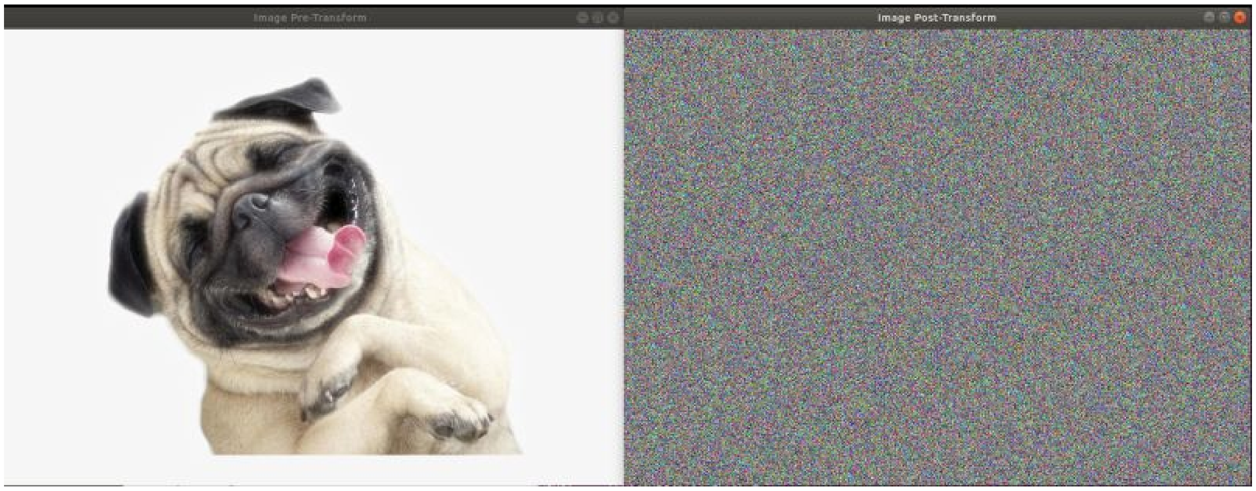
Screenshot(s) of Output:



```
osboxes@osboxes:~/COMP2040/ps1a$ ls
FibLFSR.cpp  FibLFSR.h  FibLFSR.o  Makefile  ps1a-readme.txt  test  test.cpp  test.o
osboxes@osboxes:~/COMP2040/ps1a$ ./test
Running 3 test cases...

*** No errors detected
osboxes@osboxes:~/COMP2040/ps1a$ 
```





Source Code:

Makefile:

```
CC=g++
CFLAGS=-g -Wall -Werror -ansi -pedantic -std=c++14
SFMLFLAGS=-lsfml-graphics -lsfml-window -lsfml-system
BOOSTFLAGS=-lboost_unit_test_framework
OBJ=FibLFSR.o PhotoMagic.o test.o
all: $(OBJ)
	$(CC) $(CFLAGS) -o PhotoMagic PhotoMagic.o FibLFSR.o $(SFMLFLAGS)
	$(CC) $(CFLAGS) -o test test.o FibLFSR.o $(BOOSTFLAGS)
test: test.o FibLFSR.o
	$(CC) $(CFLAGS) -o test test.o FibLFSR.o $(BOOSTFLAGS)
PhotoMagic: PhotoMagic.o FibLFSR.o
	$(CC) $(CFLAGS) PhotoMagic.o FibLFSR.o -o PhotoMagic $(SFMLFLAGS)
test.o: test.cpp
	$(CC) $(CFLAGS) -c test.cpp
PhotoMagic.o: PhotoMagic.cpp
	$(CC) $(CFLAGS) -c PhotoMagic.cpp
FibLFSR.o: FibLFSR.cpp FibLFSR.hpp
	$(CC) $(CFLAGS) -c FibLFSR.cpp
clean:
	 -@rm -rf *.o 2>/dev/null || true
```

test.cpp:

```
// Dr. Rykalova
// test.cpp for PS1a
// updated 1/31/2020
// Jacob Leboeuf 9/15/20
#include <iostream>
#include <string>

#include "FibLFSR.h"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

using namespace std;

BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
        FibLFSR l("1011011000110110");
        BOOST_REQUIRE(l.step() == 0);
        BOOST_REQUIRE(l.step() == 0);
        BOOST_REQUIRE(l.step() == 0);
        BOOST_REQUIRE(l.step() == 1);
        BOOST_REQUIRE(l.step() == 1);
        BOOST_REQUIRE(l.step() == 0);
        BOOST_REQUIRE(l.step() == 0);
        BOOST_REQUIRE(l.step() == 1);

        FibLFSR l2("1011011000110110");
        BOOST_REQUIRE(l2.generate(9) == 51);
}


/*
 * Testing operation of LSFR
 * and making sure the correct bits are generated
 * when the reg is too short to have bits in its tap positions
 */

BOOST_AUTO_TEST_CASE(noBitsInTapPosition) {
        ostringstream ss;
        FibLFSR f("110101");
        auto testCase = [&](int EXPECTED_STEP, string EXPECTED_REG_STATE) -> void
{
                BOOST_REQUIRE(f.step() == EXPECTED_STEP);
                ss << 1;
                BOOST_REQUIRE(ss.str() == EXPECTED_REG_STATE);
                ss = ostringstream{""};
        };
        testCase(1, "101011");
        testCase(1, "010111");
        testCase(0, "101110");
        testCase(1, "011101");
        testCase(0, "111010");
        FibLFSR f2("110101");
        BOOST_REQUIRE(f2.generate(5) == 26);
}

/*
 * This tests the condition of the generated bits
 * in each step of the LFSR
 * when the last bit of the reg is a tap position
 */

BOOST_AUTO_TEST_CASE(checkGenBitsWhenLastInTapPosition) {
        FibLFSR f("1011101001001");
        BOOST_REQUIRE(f.step() == 0);
        BOOST_REQUIRE(f.step() == 1);
        BOOST_REQUIRE(f.step() == 0);
        BOOST_REQUIRE(f.step() == 1);
        BOOST_REQUIRE(f.step() == 0);
        BOOST_REQUIRE(f.step() == 0);
        BOOST_REQUIRE(f.step() == 1);
        BOOST_REQUIRE(f.step() == 1);
        FibLFSR f2("1011101001001");
        BOOST_REQUIRE(f2.generate(8) == 83);
}
```

PhotoMagic.cpp:

```
/* Jacob Leboeuf
 * ps1b
 * 9/23/20
 */
#include <SFML/Graphics.hpp>
#include <iostream>
#include <algorithm>
#include "FibLFSR.h"
using namespace std;
// transforms image using FibLFSR
void transform(sf::Image& image, FibLFSR* initSeed) {
        sf::Vector2u sizeImg = image.getSize();
        sf::Color p;
        FibLFSR& f = *initSeed;
        for(size_t x = 0; x < sizeImg.x; x++) {
                for(size_t y = 0; y < sizeImg.y; y++) {
                        p = image.getPixel(x, y);
                        uint32_t rgb = p.toInteger() >> 8;      // shift 8 bits to the right
                        rgb = ((rgb ^ f.generate(24)) << 8) + 0b11111111;
                        p = sf::Color{rgb};
                        image.setPixel(x, y, p);
                }

        }
}
// display an encrypted copy of the picture, using the LFSR
// to do the encryption
int main(int argc, char* argv[]) {
        // checking if command line input was correct
        // exiting with error msg if incorrect
        if(argc < 4) {
                switch(argc) {
                        case 1:
                                cout << "No source image!" << endl;
                                exit(1);
                                break;
                        case 2:
                                cout << "No output image!" << endl;
                                exit(1);
                                break;
                        case 3:
                                cout << "No encryption seed!" << endl;
                                exit(1);
                                break;
                        }
        }
        sf::Image transformed_image;
        string output = argv[2];
        string input = argv[1];
        // error loading file
        if(!transformed_image.loadFromFile(input)) {
                cout << "Error occurred with" << input << endl;
                exit(1);
        }
        sf::Image untouched_image = transformed_image;
        FibLFSR imgSeed{string{argv[3]}};
        // call transform function
        transform(transformed_image, &imgSeed);
        transformed_image.saveToFile(output);
        sf::RenderWindow beforeTransform(sf::VideoMode(untouched_image.getSize().x,
        untouched_image.getSize().y), "Image Pre-Transform");
        sf::RenderWindow afterTransform(sf::VideoMode(transformed_image.getSize().x,
        transformed_image.getSize().y), "Image Post-Transform");
        sf::Texture preTransformTextures;
        preTransformTextures.loadFromImage(untouched_image);
        sf::Sprite preTransformSprite(preTransformTextures);
        sf::Texture postTransformTextures;
        postTransformTextures.loadFromImage(transformed_image);
        sf::Sprite postTransformSprite(postTransformTextures);
        // rules for closing / interacting with displayed images
        while (beforeTransform.isOpen() || afterTransform.isOpen()) {
                sf::Event event1, event2;
                while (beforeTransform.pollEvent(event2) || afterTransform.pollEvent(event1)) {
                        if (event1.type == sf::Event::Closed || event2.type == sf::Event::Closed
                                || sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)
                                || sf::Keyboard::isKeyPressed(sf::Keyboard::Q)) {
                                afterTransform.close();
                                beforeTransform.close();
                        }
                }
                afterTransform.clear();
                beforeTransform.clear();
                afterTransform.draw(postTransformSprite);
                beforeTransform.draw(preTransformSprite);
                afterTransform.display();
                beforeTransform.display();
        }
}
```

FibLFSR.h:

```
// Jacob Leboeuf 9/15/20
#include <iostream>
#include <string>
#include <vector>

using namespace std;
class FibLFSR {
    public:
        FibLFSR(string seed);   // constructor to create LFSR with
                                // the given initial seed and tap
        int step();             // simulate one step and return the
                                // new bit as 0 or 1
        int generate(int k);    // simulate k steps and return
                                // k-bit integer
        friend std::ostream& operator<<(ostream &out, const FibLFSR &f);

    private:
        vector<bool> bits;
};
```

FibLFSR.cpp:

```
// Jacob Leboeuf 9/15/20

#include "FibLFSR.h"

using namespace std;

FibLFSR::FibLFSR(string seed) {
    if(seed.length() == 0) {
        cout << "String can't be empty!" << endl;
    }
    for(auto p = seed.rbegin(); p != seed.rend(); p++) {
        switch(*p) {
                case '1':
                        bits.push_back(1);
                        break;
                case '0':
                        bits.push_back(0);
                        break;
                default:
                        cout << "Error" << endl;
                        exit(1);
        }
    }
}
int FibLFSR::step() {
        const int x = bits.size();
        // returns 0 if last bit is tap bit
        // returns last bit if anything else
        // returns 0 if reg too short for tap position
        bool s = (x == 14 || x == 13 || x == 11 ? 0 : bits[x - 1]) ^
                (x < 11 ? 0 : bits[10]) ^ (x < 13 ? 0 : bits[12]) ^
                (x < 14 ? 0 : bits[13]);
        bits.pop_back();
        bits.insert(bits.begin(), s);
        return s;
}
int FibLFSR::generate(int k) {
    if (k > 32) {
        cout << "Greater than 32 bits" << endl;
        exit(1);
    }
    int res = 0;
    for(int i = 0; i < k; i++) {
        res <<= 1;
        res += step();
    }
    return res;
}
std::ostream& operator<<(ostream &out, const FibLFSR &f) {
    for (auto p = f.bits.rbegin(); p != f.bits.rend(); p++) {
        out << static_cast<int>(*p);
    }
    return out;
}
```

PS2: N-Body Simulation

Assignment Description

This assignment asked us to use Newton's laws of physics to simulate movement of celestial bodies in a 2D plane within our code. The simulation created would animate the movement of the celestial bodies over time, and a movement over an interval of time would be represented by each frame of the outputted animation. The data for the celestial bodies would be read in as input, and the end-state of the N-Body system created would be outputted into a .txt file.

Key Algorithms, Data Structures, OO Design, Etc.

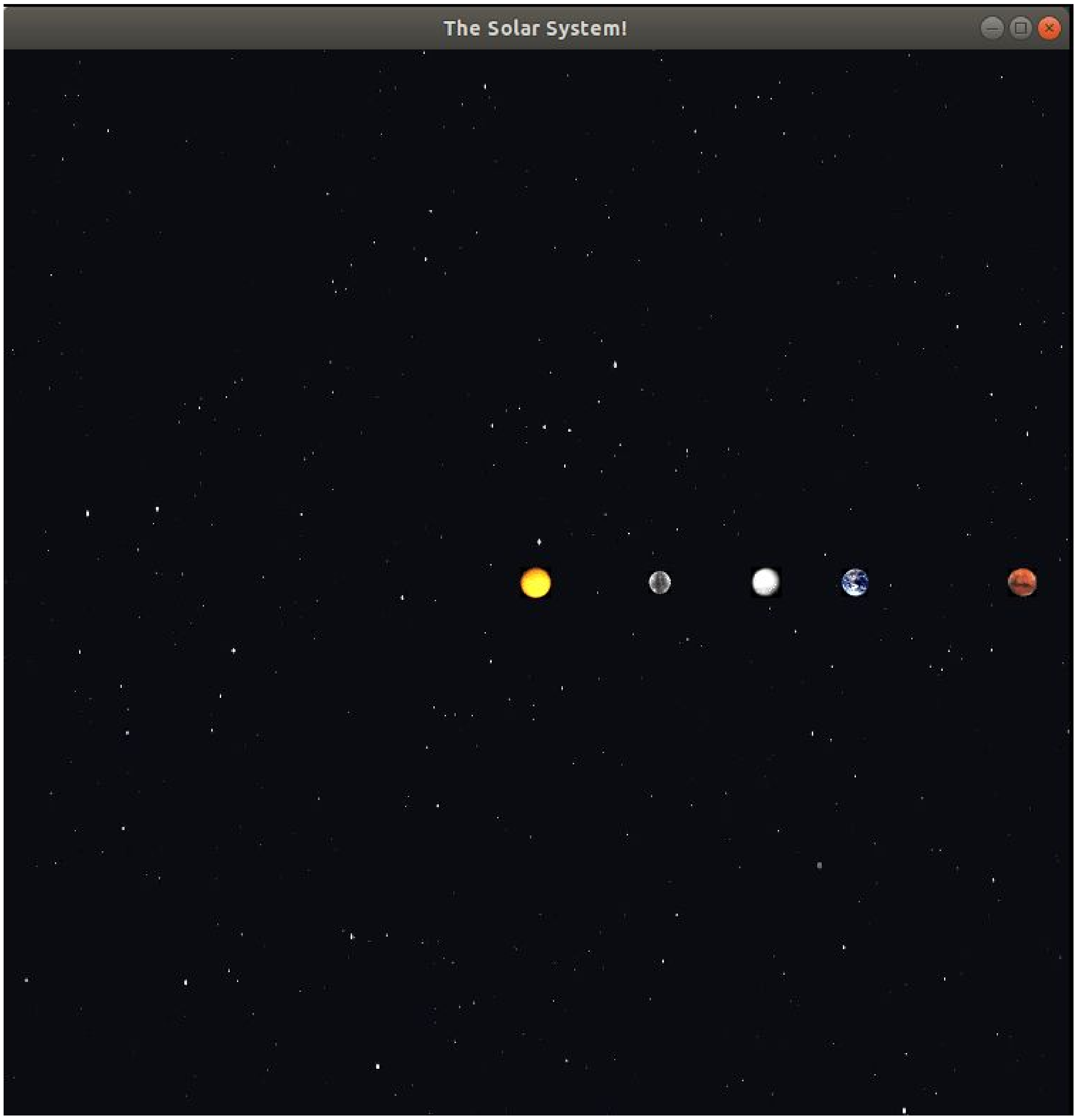
Two main classes were created in order to complete this assignment: a class that could represent each celestial body, and a "Universe" class that held all of the created celestial bodies. The CelestialBody class contained the individual body's important data, including its mass, position, and velocity, and the Universe class involved would manage and update the position of each celestial body throughout each step of the simulation.

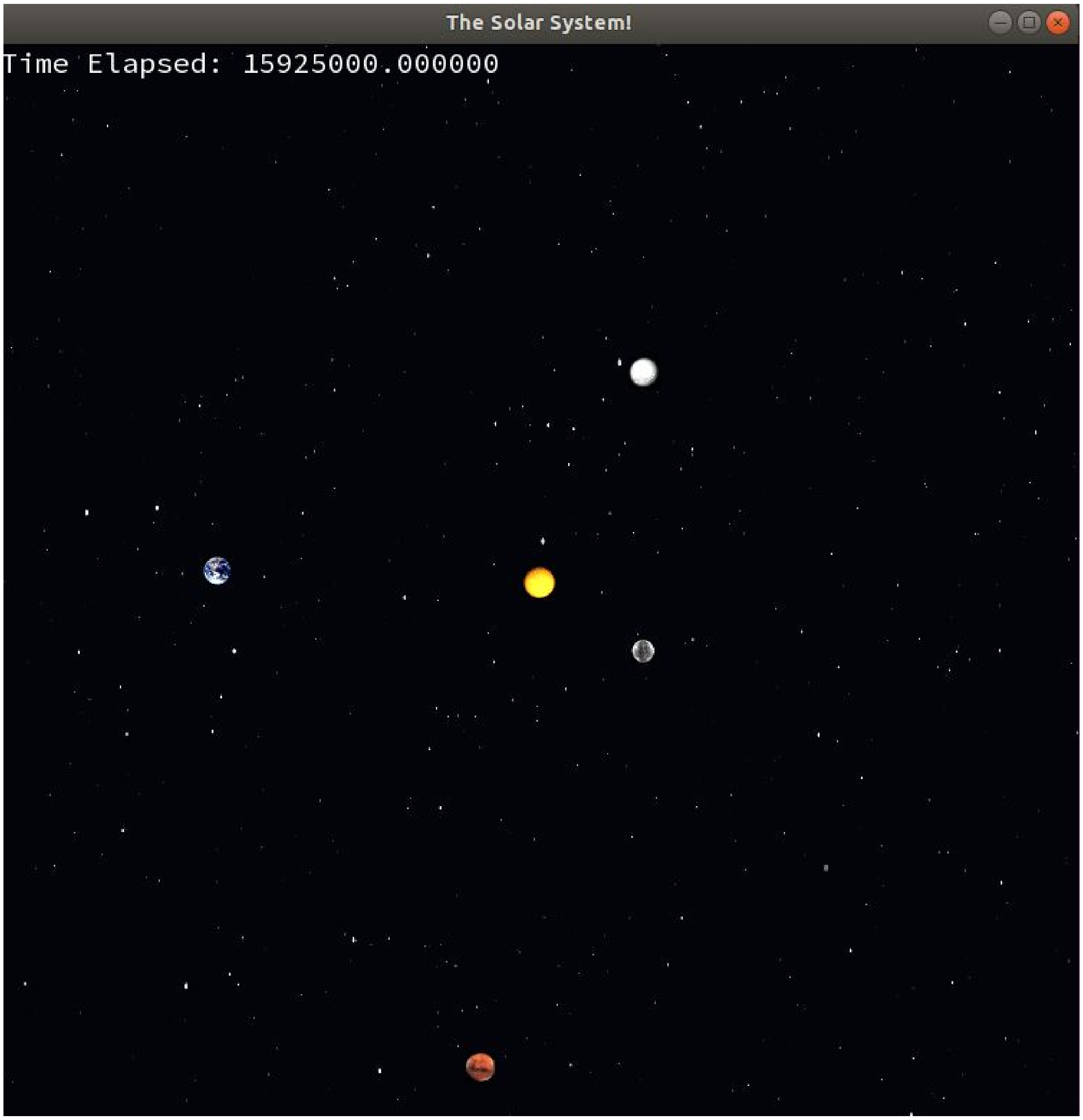
Static member functions and variables were used significantly in this assignment in order to be more efficient with data, as each CelestialBody object is referencing the same Universe center coordinate and radius. By using static member variables to represent the universe's center and radius, this ensures that only one copy of those values will be shared within the instances of the class, as opposed to having each CelestialBody hold a vector and float with the same value

What I’ve Learned

This assignment involved an introduction to smart pointers, which is a very useful feature within this programming language to know in certain situations (including the project itself). The created Universe class used smart pointers to store each celestial body within itself. On top of this, this assignment taught me efficient strategies for abstracting objects into easy to use interfaces, as shown with implementing the Universe class within the main routine.

Screenshot(s) of Output:





```
Time Elapsed: 15925000.000000
```

Source Code:

Makefile:

```
CC=g++
SFMLFLAGS=-lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
CFLAGS=-g -Wall -ansi -pedantic -std=c++14
OBJ=main.o Universe.o CelestialBody.o

all: $(OBJ)
	$(CC) $(CFLAGS) -o NBody $(OBJ) $(SFMLFLAGS)
main.o: main.cpp
	$(CC) $(CFLAGS) -c main.cpp
Universe.o: Universe.cpp Universe.hpp
	$(CC) $(CFLAGS) -c Universe.cpp
CelestialBody.o: CelestialBody.cpp CelestialBody.hpp
	$(CC) $(CFLAGS) -c CelestialBody.cpp
clean:
	-@rm -rf *.o 2>/dev/null || true
```

main.cpp:

```
#include <iostream>
#include <fstream>
#include "Universe.hpp"
#include "CelestialBody.hpp"
#include <string>
#include <exception>
#include <vector>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#define WINDOW_WIDTH 750
#define WINDOW_FPS 30
#define WINDOW_HEIGHT 750
using namespace std;
int main(int argc, char* argv[]) {
	if(argc < 2) {
		cout << "Not enough arguments!" << endl;
		return 1;
	}
	float max_time, current_time;
	float time_change;
	current_time = 0;
	try {
		max_time = stod(argv[1]);
		time_change = stod(argv[2]);
	} catch(exception e) {
		cout << "Error in args" << endl;
		return 1;
	}
	sf::Vector2f centerUniverse{WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2};
	Universe solarSystem(centerUniverse);
```

```cpp
    cin >> solarSystem;
    sf::RenderWindow window(sf::VideoMode(WINDOW_WIDTH, WINDOW_HEIGHT), "The Solar System!");
    sf::Texture spaceTextures;
    sf::Font sFont;
    sf::SoundBuffer buffer;
    window.setFramerateLimit(WINDOW_FPS);
    if(!spaceTextures.loadFromFile("spacebackground.png")) {
        throw FileNotFoundException();
        cout << "No background image selected" << endl;
    }
    sf::Sprite spaceBackground(spaceTextures);
    spaceBackground.setScale(static_cast<float>(WINDOW_WIDTH) / spaceTextures.getSize().x ,
    static_cast<float>(WINDOW_HEIGHT) / spaceTextures.getSize().y);
    if(!sFont.loadFromFile("font.ttf")) {
        throw FileNotFoundException();
    }
    if(!buffer.loadFromFile("2001.wav")) {
        throw FileNotFoundException();
    }
    sf::Sound sound(buffer);
    sf::Text timeElapsed{"Time Elapsed: " + to_string(current_time), sFont};
    timeElapsed.setPosition(0,0);
    timeElapsed.setCharacterSize(20);
    timeElapsed.setOutlineColor(sf::Color::White);
    while(window.isOpen()) {
        sf::Event event;
        while(window.pollEvent(event)) {
            sound.play();
            if (event.type == sf::Event::Closed ||
                sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
                ofstream result;
                result.open("output.txt");
                result << solarSystem;
                result.close();
                window.close();
            }
        }
        if (current_time < max_time) {
            window.clear();
            window.draw(spaceBackground);
            for(const auto &p : solarSystem.getBodies() window.draw(*p);
            window.draw(timeElapsed);
            window.display();
            solarSystem.step(time_change);
            current_time += time_change;
            timeElapsed.setString("Time Elapsed: " + to_string(current_time));
        } else {
            window.setFramerateLimit(0);
        }
    }
```

```
        return 0;
}
```

CelestialBody.hpp:

```
#ifndef CELESTIAL_BODY_HPP_
#define CELESTIAL_BODY_HPP_
#include <iostream>
#include <string>
#include <exception>
#include <iomanip>
#include <SFML/Graphics.hpp>
using namespace std;
struct FileNotFoundException : public exception {
        const char * what() const noexcept {
        return "Can't find file!";
        }
};
class CelestialBody : public sf::Drawable {
public:
        CelestialBody() {}
        CelestialBody(sf::Vector2f iPosition, sf::Vector2f iVelocity,
                float iMass, std::string iImageRef);
        inline float getMass() { return mass; }
        inline sf::Vector2f getPosition() { return position; }
        inline void setPosition(sf::Vector2f nPosition) { position = nPosition; }
        inline sf::Vector2f getVelocity() { return velocity; }
        inline void setVelocity(sf::Vector2f nVelocity) { velocity = nVelocity; }
        static void createUniverse(sf::Vector2f iCenterUniverse, float
iRadiusUniverse);
        void spriteUpdate();
        ~CelestialBody() { delete sprite_textures; };
        friend ostream& operator<<(ostream &out, const CelestialBody& cb);
private:
        sf::Vector2f position;
        sf::Vector2f velocity;
        float mass;
        string imageRef;
        sf::Sprite sprite;
        sf::Texture* sprite_textures;
        virtual void draw(sf::RenderTarget& rendTarget,
        sf::RenderStates rendStates) const;
        static sf::Vector2f centerUniverse;
        static float radiusUniverse;
};
#endif
```

CelestialBody.cpp:

```cpp
#include "CelestialBody.hpp"
using namespace std;
sf::Vector2f CelestialBody::centerUniverse{0,0};
float CelestialBody::radiusUniverse = 0;
CelestialBody::CelestialBody(sf::Vector2f iPosition, sf::Vector2f iVelocity, float iMass, string iImageRef) {
        mass = iMass;
        position = iPosition;
        velocity = iVelocity;
        imageRef = iImageRef;
        sprite_textures = new sf::Texture;
        if(!sprite_textures->loadFromFile(imageRef)) {
                throw FileNotFoundException();
        }
        sprite = sf::Sprite(*sprite_textures);
        sprite.setOrigin(sprite_textures->getSize().x / 2,
        sprite_textures->getSize().y / 2);
        spriteUpdate();
}
void CelestialBody::createUniverse(sf::Vector2f iCenterUniverse,
float iRadiusUniverse) {
        centerUniverse = iCenterUniverse;
        radiusUniverse = iRadiusUniverse;
}
void CelestialBody::spriteUpdate() {
        sf::Vector2f sprite_position{position.x / radiusUniverse * centerUniverse.x + center    Universe.x,
                position.y / radiusUniverse * centerUniverse.y + centerUniverse.y};
        sprite.setPosition(sprite_position);
}
void CelestialBody::draw(sf::RenderTarget& rendTarget, sf::RenderStates rendStates) const {
        rendTarget.draw(sprite, rendStates);
}
ostream& operator<<(ostream &out, const CelestialBody& cb) {
        out.setf(ios_base::scientific);
        out << setprecision(4) << left;
        out << setw(12) << cb.position.x << setw(12) << cb.position.y << setw(12)
                << cb.velocity.x << setw(12) << cb.velocity.y << setw(12)
                << cb.mass << right << setw(12) << cb.imageRef;
        out.unsetf(ios_base::scientific);
        return out;
}
```

Universe.hpp:

```cpp
#ifndef UNIVERSE_HPP_
#define UNIVERSE_HPP_
#include <iostream>
#include <cmath>
#include "CelestialBody.hpp"
#include <SFML/Graphics.hpp>
#include <string>
#include <vector>
using namespace std;
class Universe {
public:
        Universe() {}
        Universe(sf::Vector2f iCenter) : center(iCenter) {}
        inline const vector<unique_ptr<CelestialBody>>&
                getBodies() const { return celBodies; }
        friend istream& operator>>(istream& in, Universe& u);
        friend ostream& operator<<(ostream& out, const Universe& u);
        void step(float seconds);
private:
        sf::Vector2f center;
        float radius;
        vector<unique_ptr<CelestialBody>> celBodies;
};
# endif
```

Universe.cpp:

```cpp
#include "Universe.hpp"
using namespace std;
std::istream& operator>>(std::istream& in, Universe& u) {
        int numBodies;
        in >> numBodies >> u.radius;
        CelestialBody::createUniverse(u.center, u.radius);
        for(int i = 0; i < numBodies; i++) {
                float xPosition, yPosition, xVelocity, yVelocity, mass;
                string imageRef;
                in >> xPosition >> yPosition >> xVelocity >> yVelocity >> mass >> imageRef;
                u.celBodies.push_back(
                make_unique<CelestialBody>(sf::Vector2f(xPosition, yPosition),
                        sf::Vector2f(xVelocity, yVelocity),mass, imageRef));
        }
        return in;
}
ostream& operator<<(ostream& out, const Universe& u) {
        out << u.celBodies.size() << endl;
        out << u.radius << endl;
        for(const auto &b : u.celBodies) out << (*b) << endl;
        return out;
}
void Universe::step(float seconds) {
    auto getNetForce = [&](size_t planetIndex) -> sf::Vector2f {
        sf::Vector2f netForce;
        for(size_t i = 0; i < celBodies.size(); i++) {
            if(i != planetIndex) {
                sf::Vector2f position_change =
                    celBodies[i]->getPosition() - celBodies[planetIndex]->getPosition();
                float planetDistance = hypot(position_change.x, position_change.y);
                float scaleForce =
                    (6.67430e-11 * celBodies[planetIndex]->getMass() *
                    celBodies[i]->getMass()) / pow(planetDistance, 2);
                // 6.67430e-11 - gravitational constant
                sf::Vector2f force_xy = {
                    scaleForce * (position_change.x / planetDistance),
                    scaleForce * (position_change.y / planetDistance)
                };
                netForce += force_xy;
            }
        }
        return netForce;
    };
    vector<sf::Vector2f> rPosition, rVelocity;
    for(size_t i = 0; i < celBodies.size(); i++) {
        sf::Vector2f netForce = getNetForce(i);
        sf::Vector2f planetAccel = {
            netForce.x / celBodies[i]->getMass(),
            netForce.y / celBodies[i]->getMass()
        };
        sf::Vector2f velocity = {
            celBodies[i]->getVelocity().x + seconds * planetAccel.x,
            celBodies[i]->getVelocity().y + seconds * planetAccel.y
        };
        sf::Vector2f position = {
            celBodies[i]->getPosition().x + seconds * velocity.x,
            celBodies[i]->getPosition().y + seconds * velocity.y
        };
        rVelocity.push_back(velocity);
        rPosition.push_back(position);
    }
    for(size_t i = 0; i < celBodies.size(); i++) {
        celBodies[i]->setVelocity(rVelocity[i]);
        celBodies[i]->setPosition(rPosition[i]);
        celBodies[i]->spriteUpdate();
    }
}
```

PS3: Synthesizing a Plucked String Sound

Assignment Description

This assignment involved creating a program that would simulate sound produced similar to that of a guitar string. To do so, a Ring Buffer was implemented (named CircularBuffer), which is a queue of fixed-size which can be filled with random values. It can be iterated upon, which would delete the value in the front of the queue and enqueue a new value which is equivalent to the average of the value next in line and the deleted value, multiplied by a decay factor. This decay factor is used to simulate the "decay" of the sound from plucking a string. The Boost unit testing framework is used to ensure full, correct, and complete operation of the CircularBuffer class created. The simulation itself supports 37 keys, and can be played by using the keys on a computer keyboard.

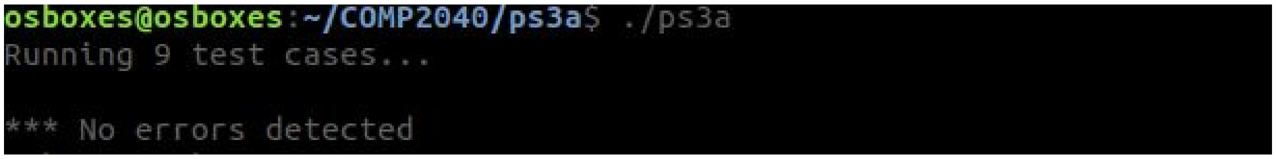
Key Algorithms, Data Structures, OO Design, Etc.

PS3 had a heavy usage of exceptions, especially within the usage of the Boost unit testing framework. Each internal function that did not simply retrieve a piece of data or returned a boolean value checked the validity of the function call based on the argument given or the state of the object when the function is being called. When these exceptions were thrown, a simple string was outputted letting the user know where the error occurred and why it did, in order to make the code more concise and easier to edit.
Lambda expressions were also used, especially to generate the samples for each of the 37 notes on the keyboard, to effectively clean up the allocated memory space when the window is eventually closed.

What I've Learned

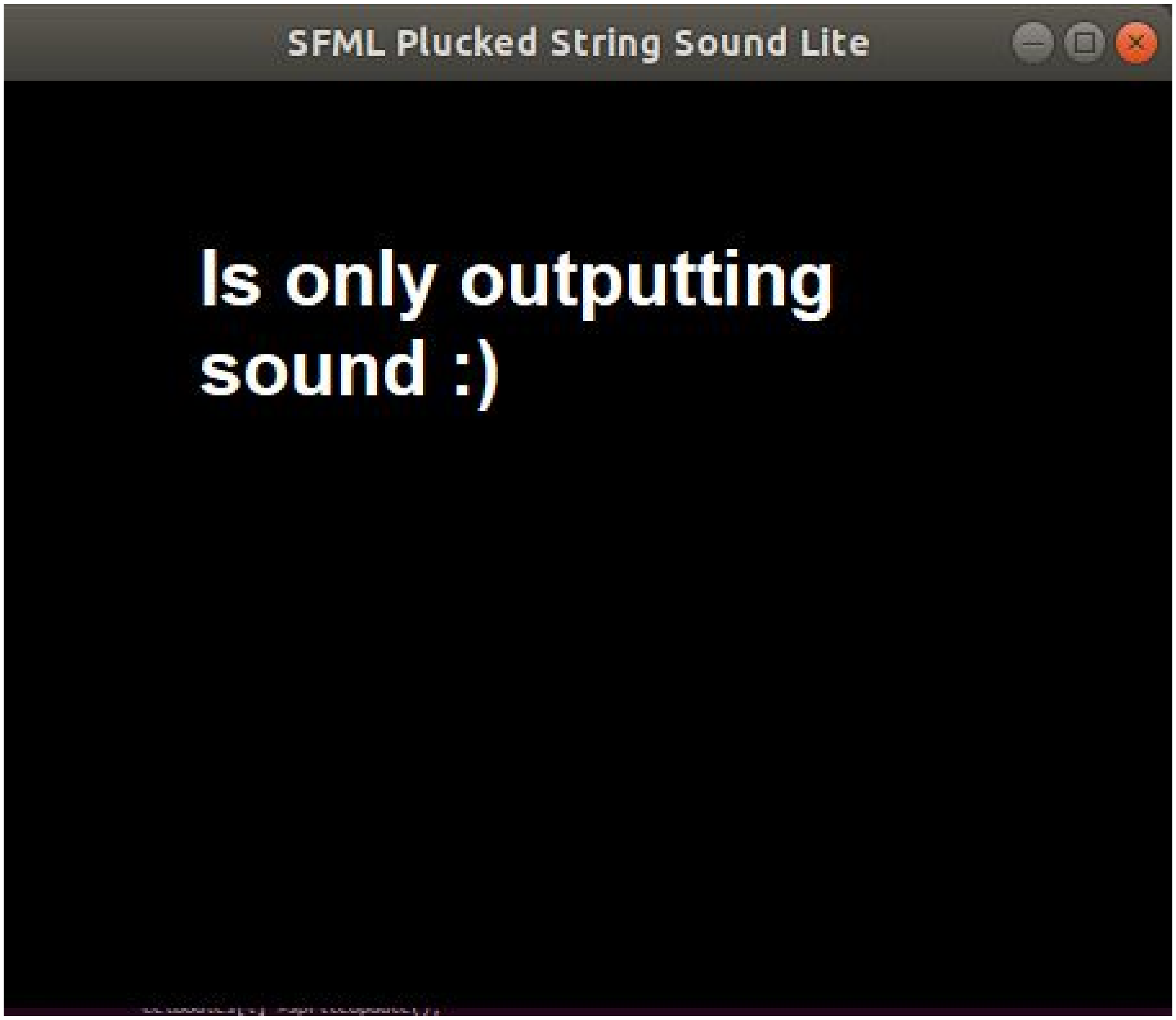
PS3 continued to help familiarize me with the Boost testing libraries and the standard exception library of C++, and become more proficient at using them. On top of this, I gained a greater understanding of how computers store and represent sound, and how you can generate the data that represent that sound using algorithms.

Screenshot(s) of Output:



```
osboxes@osboxes:~/COMP2040/ps3a$ ./ps3a
Running 9 test cases...

*** No errors detected
```



SFML Plucked String Sound Lite

Is only outputting sound :)

Source Code:

Makefile:

```
CC=g++
CFLAGS=-g -Wall -ansi -pedantic -std=c++11
OBJ= CircleBuffer.o KSGuitarSim.o
BOOSTFLAGS=-lboost_unit_test_framework
SFMLFLAGS=-lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
all: $(OBJ)
        make KSGuitarSim
        make ps3a
KSGuitarSim: CircleBuffer.o StringSound.o KSGuitarSim.o
        $(CC) $(CFLAGS) -o KSGuitarSim CircleBuffer.o StringSound.o KSGuitarSim.o
$(SFMLFLAGS)
ps3a: test.o CircleBuffer.o
        $(CC) $(CFLAGS) -o ps3a test.o CircleBuffer.o $(BOOSTFLAGS)
test.o: test.cpp
        $(CC) $(CFLAGS) -c test.cpp
CircleBuffer.o: CircleBuffer.cpp header/CircleBuffer.h
        $(CC) $(CFLAGS) -c CircleBuffer.cpp
StringSound.o: StringSound.cpp header/StringSound.h
        $(CC) $(CFLAGS) -c StringSound.cpp
KSGuitarSim.o: KSGuitarSim.cpp
        $(CC) $(CFLAGS) -c KSGuitarSim.cpp
clean:
        -@rm -rf *.o 2>/dev/null || true
```

Test.cpp:

```
// Copyright 2020 Jacob Leboeuf
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
#include <exception>
#include "header/CircularBuffer.h"
BOOST_AUTO_TEST_CASE(correctConstructorThrowsNoException) {
    BOOST_REQUIRE_NO_THROW(CircularBuffer x(50));
    BOOST_REQUIRE_NO_THROW(CircularBuffer y(999999));
    BOOST_REQUIRE_NO_THROW(CircularBuffer z(1));
}
// This tests the CircularBuffer constructor when
// its capacity is at least one and makes sure no exception is thrown
BOOST_AUTO_TEST_CASE(incorrectConstructorThrowsException) {
    BOOST_REQUIRE_THROW(CircularBuffer x(-999999), std::invalid_argument);
    BOOST_REQUIRE_THROW(CircularBuffer y(0), std::invalid_argument);
    BOOST_REQUIRE_THROW(CircularBuffer z(-1), std::invalid_argument);
}
// This tests the CircularBuffer constructor when its capacity
// is less than one and makes sure an exception is thrown

BOOST_AUTO_TEST_CASE(itemsAddedAndSizeInceasedWithEnqueue) {
    CircularBuffer c(25);
    c.enqueue(-1);
    BOOST_REQUIRE(c.size() == 1);
    c.enqueue(-2);
    BOOST_REQUIRE(c.size() == 2);
    c.enqueue(-3);
    BOOST_REQUIRE(c.size() == 3);
    for (int i = 0; i < 5; i++) {
        c.enqueue(i);
    }
    BOOST_REQUIRE(c.size() == 8);
}
// This tests the functionality of the enqueue function
// and makes sure items are added and size is increased
BOOST_AUTO_TEST_CASE(runTimeErrorThrownWithEnqueueOnFullBuffer) {
    CircularBuffer c(5);
    c.enqueue(-1);
    c.enqueue(2);
    c.enqueue(3);
    c.enqueue(4);
    c.enqueue(5);
    BOOST_REQUIRE_THROW(c.enqueue(6), std::runtime_error);
}
// This tests that a runtime error is thrown when enqueuing
// on a full buffer
BOOST_AUTO_TEST_CASE(correctValueReturnedFromDequeue) {
    CircularBuffer c(5);
    c.enqueue(1);
    BOOST_REQUIRE(c.dequeue() == 1);
    BOOST_REQUIRE(c.size() == 0);
    c.enqueue(2);
    c.enqueue(3);
    c.enqueue(4);
    BOOST_REQUIRE(c.dequeue() == 2);
    BOOST_REQUIRE(c.dequeue() == 3);
    c.enqueue(5);
    BOOST_REQUIRE(c.dequeue() == 4);
    BOOST_REQUIRE(c.dequeue() == 5);
}
// This tests that the correct value in the buffer is
// returned from dequeue

BOOST_AUTO_TEST_CASE(ExceptionThrownWhenDequeueingOnEmptyBuffer) {
    CircularBuffer c(3);
    c.enqueue(1);
    c.enqueue(2);
    c.enqueue(3);
    c.dequeue();
    c.dequeue();
    c.dequeue();
    BOOST_REQUIRE_THROW(c.dequeue(), std::runtime_error);
}
// This tests if a runtime error is thrown when dequeueing
// on an empty buffer
BOOST_AUTO_TEST_CASE(ItemsRemovedandSizeDecreasedFromDequeue) {
    CircularBuffer c(4);
    c.enqueue(1);
    c.dequeue();
    BOOST_REQUIRE(c.size() == 0);
    c.enqueue(2);
    c.enqueue(3);
    c.enqueue(4);
    c.enqueue(5);
    c.dequeue();
    BOOST_REQUIRE(c.size() == 3);
    c.dequeue();
    c.dequeue();
    BOOST_REQUIRE(c.size() == 1);
}
// This tests the dequeue function
// and its ability to decrease the size and remove the correct item
BOOST_AUTO_TEST_CASE(peekReturnsCorrectValue) {
    CircularBuffer c(4);
    c.enqueue(1);
    BOOST_REQUIRE(c.peek() == 1);
    c.enqueue(2);
    BOOST_REQUIRE(c.peek() == 1);
    c.dequeue();
    BOOST_REQUIRE(c.peek() == 2);
    c.dequeue();
    c.enqueue(3);
    BOOST_REQUIRE(c.peek() == 3);
}
// This tests if peek returns the correct value in the buffer
BOOST_AUTO_TEST_CASE(exceptionThrownWhenPeekingAtEmptyBuffer) {
    CircularBuffer c(3);
    c.enqueue(1);
    c.enqueue(2);
    c.enqueue(3);
    c.dequeue();
    c.dequeue();
    c.dequeue();
    BOOST_REQUIRE_THROW(c.peek(), std::runtime_error);
}
// This tests if an exception is thrown
// when peeking at an empty buffer
```

KSGuitarSim.cpp:

```cpp
/*
  Copyright 2015 Fred Martin,
  Y. Rykalova, 2020
  Jacob Leboeuf, 2020
*/

#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>
#include <math.h>

#include <limits.h>
#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include <vector>

#include "header/CircleBuffer.h"
#include "header/StringSound.h"

int main() {
        const std::vector<sf::Keyboard::Key> KEYSTRINGS = {
                sf::Keyboard::Q, sf::Keyboard::Num2, sf::Keyboard::W, sf::Keyboard::E,
                sf::Keyboard::Num4, sf::Keyboard::R, sf::Keyboard::Num5, sf::Keyboard::T,
                sf::Keyboard::Y, sf::Keyboard::Num7, sf::Keyboard::U, sf::Keyboard::Num8,
                sf::Keyboard::I, sf::Keyboard::Num9, sf::Keyboard::O, sf::Keyboard::P,
                sf::Keyboard::Dash, sf::Keyboard::LBracket, sf::Keyboard::Equal,
                sf::Keyboard::Z, sf::Keyboard::X, sf::Keyboard::D, sf::Keyboard::C,
                sf::Keyboard::F, sf::Keyboard::V, sf::Keyboard::G, sf::Keyboard::B,
                sf::Keyboard::N, sf::Keyboard::J, sf::Keyboard::M, sf::Keyboard::K,
                sf::Keyboard::Comma, sf::Keyboard::Period, sf::Keyboard::SemiColon,
                sf::Keyboard::Slash, sf::Keyboard::Quote, sf::Keyboard::Space
        };
        auto makeSamples = [&] (StringSound gs) -> std::vector<sf::Int16> {
                std::vector<sf::Int16> samples;
                gs.pluck();
                int duration = 8;  // seconds
                int i;
                for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
                        gs.tic();
                        samples.push_back(gs.sample());
                 }

        return samples;
    };
    std::vector<std::vector<sf::Int16>> samples;
    std::vector<sf::SoundBuffer*> sBufferArr;
    std::vector<sf::Sound*> soundArr;
    sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Plucked String Sound Lite");
    sf::Event event;
    for(int i = 1; i <= 37; i++) {
        double freq = CONCERT_A * std::pow(2, (i - 24) / 12.0);
        std::vector<sf::Int16> madeSamples = makeSamples(StringSound(freq));
        samples.push_back(madeSamples);
        sf::SoundBuffer* nBuf = new sf::SoundBuffer();
        nBuf->loadFromSamples(madeSamples.data(), madeSamples.size(), 1, SAMPLES_PER_SEC);
        sBufferArr.push_back(nBuf);
        sf::Sound* nSound = new sf::Sound();
        nSound->setBuffer(*nBuf);
        soundArr.push_back(nSound);
    }
    auto makeClean = [&]() -> void {
        for(auto &a : sBufferArr) delete a;
        for(auto &a : soundArr) delete a;
    };
    while(window.isOpen()) {
        while (window.pollEvent(event)) {
            if(event.type == sf::Event::Closed) {
                makeClean();
                window.close();
            }
            if(event.type == sf::Event::KeyPressed) {
                auto currKey = std::find(KEYSTRINGS.begin(), KEYSTRINGS.end(), event.key.code);
                if(currKey == KEYSTRINGS.end()) {
                    if(event.key.code == sf::Keyboard::Escape) {
                        makeClean();
                        window.close();
                    }
                }
                else {
                    int keyIndex = std::distance(KEYSTRINGS.begin(), currKey);
                    soundArr[keyIndex]->play();
                }
            }
        }
        window.clear();
        window.display();
    }
    return 0;
}
```

CircleBuffer.h:

```
// Copyright 2020 Jacob Leboeuf
// NOLINTNEXTLINE
#ifndef _HEADER_CIRCLEBUFFER_H_ // NOLINT
#define _HEADER_CIRCLEBUFFER_H_
#include <cstdint>
#include <exception>
#include <vector>
class CircleBuffer {
 public:
     explicit CircleBuffer(int _capacity);
     // create an empty ring buffer, with given max capacity
     inline int size() { return queue.size(); }
     // return number of items currently in the buffer
     inline bool isEmpty() { return queue.size() == 0; }
     // is the buffer empty (size equals zero)?
     inline bool isFull() { return queue.size() == capacity; }
     // is the buffer full (size equals capacity)?
     void enqueue(int16_t x);
     // add item x to the end
     int16_t dequeue();
     // delete and return item from the front
     int16_t peek();
     // return (but do not delete) item from the front
     void empty();
 private:
     std::vector<int16_t> queue;
     size_t capacity;
};
#endif  // _HEADER_CIRCLEBUFFER_H_
```

CircleBuffer.cpp:

```
// Copyright 2020 Jacob Leboeuf
// NOLINTNEXTLINE
#include "header/CircleBuffer.h" // NOLINT
#include <iostream>
#include <vector>
CircleBuffer::CircleBuffer(int _capacity) {
     if (_capacity < 1) {
          throw std::invalid_argument(
            "CircleBuffer constructor: capaacity must be greater than zero");
     }
     capacity = _capacity;
     queue.reserve(capacity);
}
// is the buffer full (size equals capacity)?
void CircleBuffer::enqueue(int16_t x) {
     if (queue.size() == capacity) {
          throw std::runtime_error(
               "enqueue: can't enqueue to a full ring");
     }
     queue.push_back(x);
}
// add item x to the end
int16_t CircleBuffer::dequeue() {
     if (queue.size() == 0) {
          throw std::runtime_error(
               "Error! Buffer is empty! Can't dequeue!");
     }
     int16_t i = queue[0];
     queue.erase(queue.begin(), queue.begin() + 1);
     return i;
}
// delete and return item from the front
int16_t CircleBuffer::peek() {
     if (queue.size() == 0) {
          throw std::runtime_error(
              "Error! Buffer is empty! Can't peek!");
     }
     return queue[0];
}
// return (but do not delete) item from the front
void CircleBuffer::empty() {
     if (!isEmpty()) {
         std::vector<int16_t> v;
         v.reserve(capacity);
         queue = v;
     }
}
// clear/empty CircularBuffer queue
```

StringSound.h:

```
// Copyright Jacob Leboeuf 2020
#include <exception>
#include <SFML/Graphics.hpp>
#include <vector>
#include <cmath>
#include "CircleBuffer.h"
#define CONCERT_A 440.0
#define SAMPLES_PER_SEC 44100
class StringSound {
public:
        StringSound(double frequency);    // create a guitar string sound of the
                                          // given frequency using a sampling rate
                                                                    // of 44,100
        StringSound(std::vector<sf::Int16> init);   // create a guitar string with
                                           // size and initial values are given by
                                                                   // the vector
        void pluck();                      // pluck the guitar string by replacing
                                                 // the buffer with random values,
                                                       // representing white noise
        void tic();                        // advance the simulation one time step
        inline sf::Int16 sample() { return buffer->peek(); } // return the current sample
        inline int time() { return ticTimes; } // return the number of times tic was called
                                                                            // so far
        ~StringSound();                                                 // destructor
private:
        int ticTimes;
        CircleBuffer* buffer;
};
```

StringSound.cpp:

```
// Copyright Jacob Leboeuf 2020
#include <iostream>
#include "header/StringSound.h"
using namespace std;
// create a guitar string sound of the
// given frequency using a sampling rate
// of 44,100
StringSound::StringSound(double frequency) {
        if (frequency <= 0) {
                throw invalid_argument("Error! Frequency can't be 0 or less!");
        }
        int cap = ceil(SAMPLES_PER_SEC / frequency);
        buffer = new CircleBuffer(cap);
        ticTimes = 0;
}
// create a guitar string with
// size and initial values are given by
// the vector
StringSound::StringSound(std::vector<sf::Int16> init) {
        if (init.size() == 0) {
                throw std::invalid_argument("Error! Initial list is empty!");
        }
        buffer = new CircleBuffer(init.size());
        for(auto &a : init)
                buffer->enqueue(a);
}
// pluck the guitar string by replacing
// the buffer with random values,
// representing white noise
void StringSound::pluck() {
        buffer->empty();
        while(!buffer->isFull()) {
                int16_t randomNum = rand() % (32768 + 32767) - 32768;
                // Range: -32768 to 32767
                // Low bound: -32768
                buffer->enqueue(randomNum);
        }
        ticTimes++;
}
// advance the simulation one time step
```

```
void StringSound::tic() {
        int16_t removedSample = buffer->dequeue();
        int16_t beginSample = buffer->peek();
        int16_t nextSample = ceil(0.5 * (beginSample + removedSample) * 0.996);
        // Factor at which frequency is "decayed": 0.996
        buffer->enqueue(nextSample);
        ticTimes++;
}
// destructor
StringSound::~StringSound() {
        delete buffer;
}
```

PS4: DNA Sequence Alignment

Assignment Description

This assignment involved creating a class that could compute the edit distance between two strings, weighting such distance to simulate mutations that are possible when aligning sequences of DNA. To perform such a task, a class must be created that can accept two strings, calculate said edit distance between the two of them, and compute the optimal set of operations that could be performed in order to have both strings be equal to one another. Such operations include insertion, deletion, and substitution.

Key Algorithms, Data Structures, OO Design, Etc.

PS4 required the usage of a two-dimensional array to hold and contain each possible edit distance between two strings for each set of operations. This dynamic programming approach was implemented with a vector, containing another vector of fast unsigned 32 bit integers. To avoid extraneous resizes and making the program run slower, each vector would only be resized once when the 2D array was being generated. This was to account for the assignment's excessive stress on the speed of the operations being performed to complete the desired task.

What I’ve Learned

I may not have been able to fully complete the assignment at hand, but overall it still left me with plenty of takeaways. This assignment helped me better understand various approaches to optimize code when the speed at which it is run is one of the main factors. On top of this, working with two-dimensional arrays helped me gain knowledge and experience with working with multi-dimensional data structures in general, as I did not have significant prior experience in doing so.

Screenshot(s) of Output:

None, as unfortunately the assignment itself was not fully completed.

Source Code:

Makefile:

```
CC=g++
CFLAGS=-g -Wall -ansi -pedantic -std=c++11
SFMLFLAGS=-lsfml-system
OBJ=EditDistance.o main.o
all: $(OBJ)
	make ps4
ps4: $(OBJ)
	$(CC) $(CFLAGS) -o ps4 main.o EditDistance.o $(SFMLFLAGS)
EditDistance.o: EditDistance.cpp header/EditDistance.h
	$(CC) $(CFLAGS) -c EditDistance.cpp
main.o: main.cpp
	$(CC) $(CFLAGS) -c main.cpp
clean:
	-@rm -rf *.o 2>/dev/null || true
```

EditDistance.h:

```
// Copyright 2020 Jacob Leboeuf
// NOLINTNEXTLINE
#ifndef _HEADER_EDITDISTANCE_H_  // NOLINT
#define _HEADER_EDITDISTANCE_H_
#include <iostream>
#include <cstdint>
#include <vector>
#include <algorithm>
#include <string>
class EditDistance {
 public:
      EditDistance(std::string s1, std::string s2);
      std::string Alignment();
      unsigned int OptDistance();
 private:
      int x;
      int y;
      std::vector<std::vector<uint_fast32_t>> opt;
      static inline int min(int a, int b, int c) {
            return a < b ? ( a < c ? a : c) : (b < c ? b : c);
      }
      static int penalty(char a, char b);
      void print();
};
// NOLINTNEXTLINE
#endif  // _HEADER_EDITDISTANCE_H_
```

EditDistance.cpp:

```
// Copyright Jacob Leboeuf 2020

#include <iostream>
#include <string>
#include "header/EditDistance.h"
EditDistance::EditDistance(std::string s1, std::string s2) {
     s1.push_back('-');  // load in dash first to prevent future errors
     s2.push_back('-');
     const size_t LENGTH_S1 = s1.length();
     const size_t LENGTH_S2 = s2.length();
     opt.resize(LENGTH_S1);
     for (size_t i = 0; i < LENGTH_S1; i++) {
          opt[i].resize(LENGTH_S2);
          opt[i][LENGTH_S2 - 1] = ((LENGTH_S1 - 1 - i) << 1);
     }
     for (size_t j = 0; j < LENGTH_S2; j++) {
          opt[LENGTH_S1 - 1][j] = ((LENGTH_S2 - 1 - j) << 1);
     }
     EditDistance::print();
     for (size_t i = 0; i < LENGTH_S1; i++) {
          for (size_t j = 0; j < LENGTH_S2; j++) {
               bool substitutionNeeded = s1[i + 1] == s2[j + 1];
          }
     }
}
std::string EditDistance::Alignment() {
     return std::string();
}
unsigned int EditDistance::OptDistance() {
     return 0;
}
int EditDistance::penalty(char a, char b) {
     if (a == b) {
          return 0;
     }
     if (a == '-' || b == '-') {
          return 2;
     }
     return 1;
}
void EditDistance::print() {
     for (auto &o : opt) {
          for (auto &iO : o) {
               std::cout << iO << " ";
          }
          std::cout << std::endl;
     }
     std::cout << std::endl;
}
```

PS5: Markov Model of Natural Language

Assignment Description

This assignment involved implementing a Markov model to analyze transitions between units of k characters, or k-grams, within a given text. Using this model, PS5 asked us to generate text probabilistically based off of what was being fed to the model itself. In order to accomplish this, an arbitrary length of random text was generated based off of the input text and the results of reading that text from the model class. The assignment also used the Boost testing unit framework to ensure the complete and correct operability of the MModel class created.

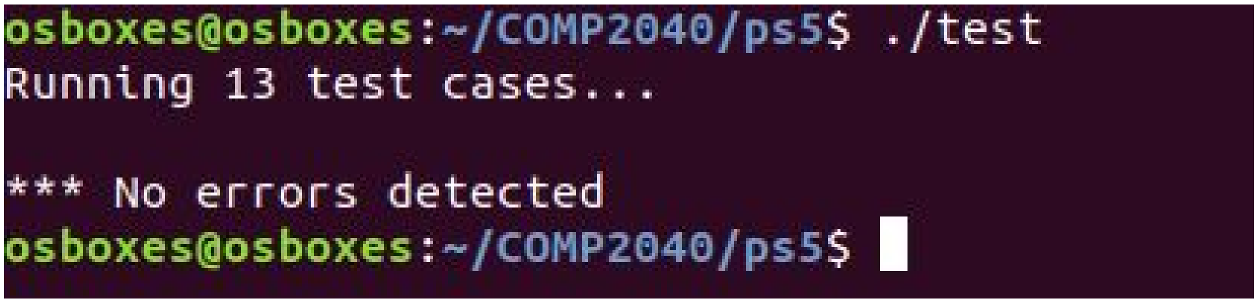
Key Algorithms, Data Structures, OO Design, Etc.

The map object of C++ was essential to the functionality of the entire assignment and of the MModel class. Specifically, what was used was a map with a key-value pair of a string and another nested map, containing a key-value pair of a character and unsigned integer.
The created map would pair the k-gram string with a second map that contains an entry for every character that proceeds said k-gram, and an unsigned integer that represents the number of times that character would do so. The usage of the map object allows me to avoid dynamically allocating memory and provides an easy visualization of the hierarchical structure of the model's data.

What I've Learned

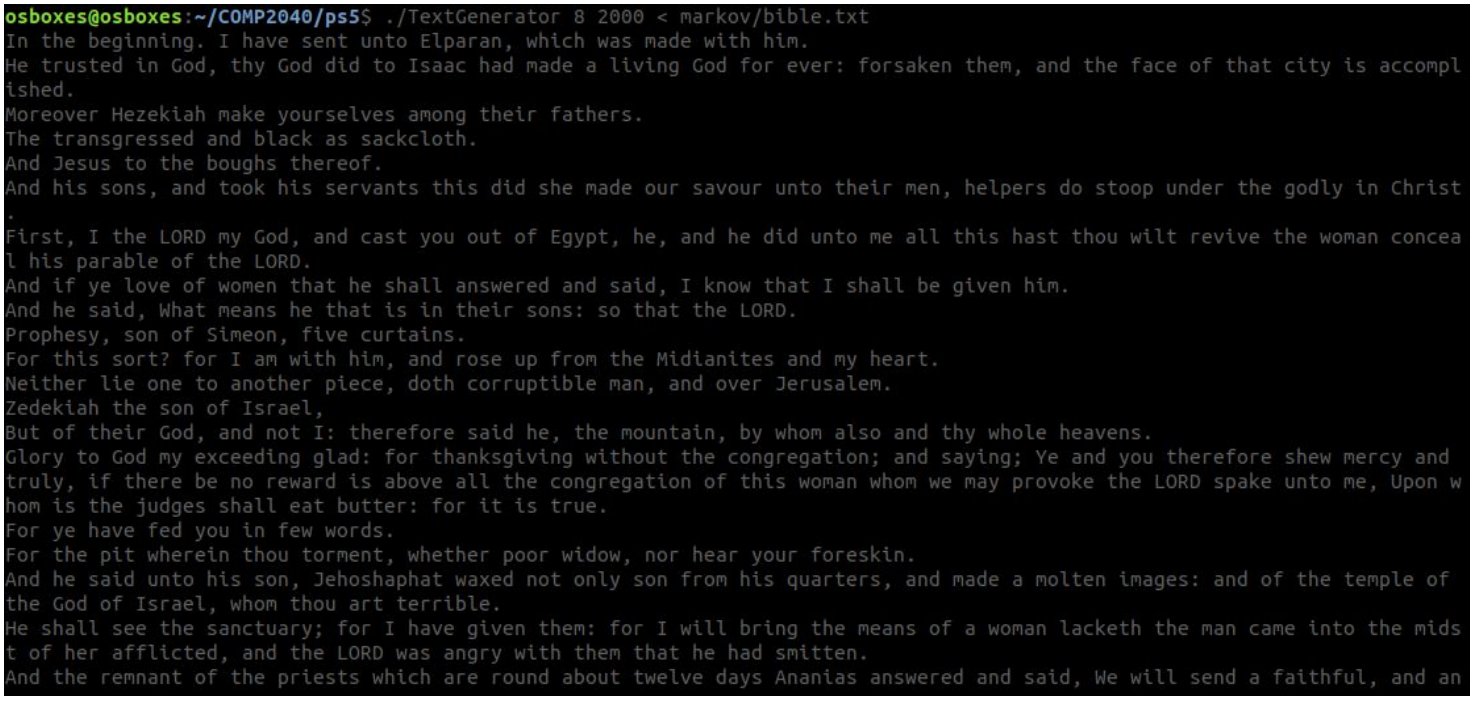
Considering that this was very likely the most difficult project I have worked on that included the map object, I would say that this assignment has improved my understanding of key-value data structures as well as complex data structures in general. This better understanding includes how they can be traversed, and more experience with accessing the data within them.

Screenshot(s) of Output:



```
osboxes@osboxes:~/COMP2040/ps5$ ./test
Running 13 test cases...

*** No errors detected
osboxes@osboxes:~/COMP2040/ps5$
```



```
osboxes@osboxes:~/COMP2040/ps5$ ./TextGenerator 8 2000 < markov/bible.txt
In the beginning. I have sent unto Elparan, which was made with him.
He trusted in God, thy God did to Isaac had made a living God for ever: forsaken them, and the face of that city is accompl
ished.
Moreover Hezekiah make yourselves among their fathers.
The transgressed and black as sackcloth.
And Jesus to the boughs thereof.
And his sons, and took his servants this did she made our savour unto their men, helpers do stoop under the godly in Christ
.
First, I the LORD my God, and cast you out of Egypt, he, and he did unto me all this hast thou wilt revive the woman concea
l his parable of the LORD.
And if ye love of women that he shall answered and said, I know that I shall be given him.
And he said, What means he that is in their sons: so that the LORD.
Prophesy, son of Simeon, five curtains.
For this sort? for I am with him, and rose up from the Midianites and my heart.
Neither lie one to another piece, doth corruptible man, and over Jerusalem.
Zedekiah the son of Israel,
But of their God, and not I: therefore said he, the mountain, by whom also and thy whole heavens.
Glory to God my exceeding glad: for thanksgiving without the congregation; and saying; Ye and you therefore shew mercy and
truly, if there be no reward is above all the congregation of this woman whom we may provoke the LORD spake unto me, Upon w
hom is the judges shall eat butter: for it is true.
For ye have fed you in few words.
For the pit wherein thou torment, whether poor widow, nor hear your foreskin.
And he said unto his son, Jehoshaphat waxed not only son from his quarters, and made a molten images: and of the temple of
the God of Israel, whom thou art terrible.
He shall see the sanctuary; for I have given them: for I will bring the means of a woman lacketh the man came into the mids
t of her afflicted, and the LORD was angry with them that he had smitten.
And the remnant of the priests which are round about twelve days Ananias answered and said, We will send a faithful, and an
```

Source Code:

Makefile:

```
CC=g++
OBJ=MModel.o TextGenerator.o test.o
CFLAGS=-g -Wall -ansi -pedantic -std=c++11
BOOSTFLAGS=-lboost_unit_test_framework
all: $(OBJ)
        $(CC) $(CFLAGS) -o TextGenerator TextGenerator.o MModel.o
        $(CC) $(CFLAGS) -o test test.o MModel.o $(BOOSTFLAGS)
TextGenerator: TextGenerator.o MModel.o
        $(CC) $(CFLAGS) -o TextGenerator TextGenerator.o MModel.o
test: test.o
        $(CC) $(CFLAGS) -o test test.o MModel.o $(BOOSTFLAGS)
MModel.o: MModel.cpp header/MModel.h
        $(CC) $(CFLAGS) -c MModel.cpp
test.o: test.cpp
        $(CC) $(CFLAGS) -c test.cpp
TextGenerator.o: TextGenerator.cpp
        $(CC) $(CFLAGS) -c TextGenerator.cpp
clean:
        -@rm -rf *.o 2>/dev/null || true
```

Test.cpp:

```
// Copyright Jacob Leboeuf 2020
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
#include <iostream>
#include <string>
#include "header/MModel.h"
BOOST_AUTO_TEST_CASE(testConstructorWithStringLessThanK) {
     BOOST_REQUIRE_THROW(MModel("abcd", 5), std::invalid_argument);
     BOOST_REQUIRE_THROW(MModel("abc", 25), std::invalid_argument);
     BOOST_REQUIRE_NO_THROW(MModel("abcd", 4));
}
BOOST_AUTO_TEST_CASE(testingFunctionalityOfKOrder) {
     MModel m1("gagggagaggcgagaaa", 5);
     MModel m2("gagggagaggcgagaaa", 10);
     MModel m3("gagggagaggcgagaaa", 15);
     BOOST_REQUIRE(m1.kOrder() == 5);
     BOOST_REQUIRE(m2.kOrder() == 10);
     BOOST_REQUIRE(m3.kOrder() == 15);
}
BOOST_AUTO_TEST_CASE(testFreqWithKGramOfInvalidLength) {
     MModel m1("gagggagaggcgagaaa", 2);
     BOOST_REQUIRE_THROW(m1.freq("a", 'c'), std::invalid_argument);
     BOOST_REQUIRE_THROW(m1.freq("gac"), std::invalid_argument);
     BOOST_REQUIRE_THROW(m1.freq("a"), std::invalid_argument);
     BOOST_REQUIRE_THROW(m1.freq("gac", 'c'), std::invalid_argument);
     BOOST_REQUIRE_NO_THROW(m1.freq("ga", 'c'));
     BOOST_REQUIRE_NO_THROW(m1.freq("ga"));
}
BOOST_AUTO_TEST_CASE(testIfInputStringIsParsedCircularly) {
     MModel m1("gagggagaggcgagaaa", 2);
     MModel m2("abcdefghijklmnop", 5);
     MModel m3("aacabbacb", 2);
     BOOST_REQUIRE(m1.freq("ag") == 5);
     BOOST_REQUIRE(m2.freq("pabcd") == 1);
     BOOST_REQUIRE(m2.freq("opabc") == 1);
     BOOST_REQUIRE(m2.freq("nopab") == 1);
     BOOST_REQUIRE(m2.freq("mnopa") == 1);
     BOOST_REQUIRE(m3.freq("ba") == 2);
}
BOOST_AUTO_TEST_CASE(testingFunctionalityOfKRand) {
     MModel m1("gagggagaggcgagaaa", 2);
     char c = m1.kRand("ga");
     BOOST_REQUIRE(c == 'a' || c == 'g');
     c = m1.kRand("gc");
     BOOST_REQUIRE(c == 'g');
     c = m1.kRand("cg");
     BOOST_REQUIRE(c == 'a');
     c = m1.kRand("gg");
     BOOST_REQUIRE(c == 'a' || c == 'c' || c == 'g');
}
BOOST_AUTO_TEST_CASE(testingKRandWithKGramOfInvalidLength) {
     MModel m1("gagggagaggcgagaaa", 2);
     BOOST_REQUIRE_THROW(m1.kRand("a"), std::invalid_argument);
     BOOST_REQUIRE_THROW(m1.kRand("gac"), std::invalid_argument);
     BOOST_REQUIRE_NO_THROW(m1.kRand("ga"));
}
BOOST_AUTO_TEST_CASE(testingkRandWithKGramNotPresentInKOrder) {
     MModel m1("gagggagaggcgagaaa", 2);
     BOOST_REQUIRE_THROW(m1.kRand("ac"), std::runtime_error);
     BOOST_REQUIRE_THROW(m1.kRand("ca"), std::runtime_error);
     BOOST_REQUIRE_NO_THROW(m1.kRand("aa"));
     BOOST_REQUIRE_NO_THROW(m1.kRand("ga"));
}
BOOST_AUTO_TEST_CASE(testingGenerateWithKGramNotPresentInKOrder) {
     MModel m1("gagggagaggcgagaaa", 2);
     BOOST_REQUIRE_THROW(m1.generate("cc", 5), std::runtime_error);
     BOOST_REQUIRE_THROW(m1.generate("ac", 9), std::runtime_error);
     BOOST_REQUIRE_NO_THROW(m1.generate("gg", 19));
     BOOST_REQUIRE_NO_THROW(m1.generate("aa", 49));
}
BOOST_AUTO_TEST_CASE(testingFunctionalityOfGenerate) {
     MModel m1("gagggagaggcgagaaa", 2);
     BOOST_REQUIRE(m1.generate("ga", 2).length() == 2);
     BOOST_REQUIRE(m1.generate("ga", 3).length() == 3);
     BOOST_REQUIRE(m1.generate("ga", 50).length() == 50);
}
BOOST_AUTO_TEST_CASE(generateWithLengthOfKgramReturnsKgram) {
     MModel m1("gagggagaggcgagaaa", 5);
     BOOST_REQUIRE(m1.generate("aggcg", 5) == "aggcg");
     BOOST_REQUIRE(m1.generate("gaggg", 5) == "gaggg");
```

```
}
BOOST_AUTO_TEST_CASE(testingGenerateWithKGramOfInvalidLength) {
     MModel m1("gagggagaggcgagaaa", 2);
     BOOST_REQUIRE_THROW(m1.generate("a", 40), std::invalid_argument);
     BOOST_REQUIRE_THROW(m1.generate("gac", 40), std::invalid_argument);
     BOOST_REQUIRE_NO_THROW(m1.generate("ga", 40));
}
BOOST_AUTO_TEST_CASE(testingGenerateWithInputStringOfInvalidLength) {
     MModel m1("gagggagaggcgagaaa", 4);
     MModel m2("gagggagaggcgagaaa", 2);
     BOOST_REQUIRE_THROW(m1.generate("gagg", 3), std::invalid_argument);
     BOOST_REQUIRE_NO_THROW(m1.generate("gagg", 22));
     BOOST_REQUIRE_NO_THROW(m1.generate("gagg", 4));
     BOOST_REQUIRE_THROW(m2.generate("ga", 1), std::invalid_argument);
     BOOST_REQUIRE_NO_THROW(m2.generate("ga", 6));
     BOOST_REQUIRE_NO_THROW(m2.generate("ga", 2));
}
BOOST_AUTO_TEST_CASE(outputOperatorReturnsOstreamRef) {
     MModel m1("gagggagaggcgagaaa", 2);
     std::cout.setstate(std::ios_base::badbit);
     BOOST_REQUIRE(typeid(std::cout << m1) == typeid(std::cout));
}
```

MModel.h:

```
// Copyright Jacob Leboeuf 2020
// NOLINTNEXTLINE
#ifndef HEADER_MMODEL_H_ // NOLINT
#define HEADER_MMODEL_H_
#include <iostream>
#include <string>
#include <map>
class MModel {
 public:
     // Note: all of the below constructors/methods should be public.
     // create a Markov model of order k from given text
     MModel(std::string text, unsigned int k);
     // Assume that text has length at least k.
     inline unsigned int kOrder() { return order; }  // order k of Markov model
     // number of occurrences of kgram in text
     int freq(std::string kgram) const;
     // (throw an exception if kgram is not of length k)
     // number of times that character c follows kgram
     // if order=0, return num of times char c appears
     // (throw an exception if kgram is not of length k)
     int freq(std::string kgram, char c) const;
     // random character following given kgram
     // (Throw an exception if kgram is not of length k.
     // Throw an exception if no such kgram.)
     char kRand(std::string kgram);
     //  generate a string of length L characters
     //  by simulating a trajectory through the corresponding
     //  Markov chain. The first k characters of the newly
     //  generated string should be the argument kgram.
     //  Throw an exception if kgram is not of length k.
     //  Assume that L is at least k.
     std::string generate(std::string kgram, unsigned int L);
     //  overload the stream insertion operator and display
     //  the internal state of the Markov Model. Print out
     //  the order, the alphabet, and the frequencies of
     //  the k-grams and k+1-grams.
     friend std::ostream& operator<<(std::ostream& out, const MModel& m);

 private:
     std::map<std::string, std::map<char, unsigned int>> markovMaps;
     unsigned int order;
     std::string alphabet;
};
// NOLINTNEXTLINE
#endif  // HEADER_MMODEL_H_
```

MModel.cpp:

```
// Copyright Jacob Leboeuf 2020
#include "header/MModel.h"
#include <cstdlib>
#include <algorithm>
#include <string>
MModel::MModel(std::string text, unsigned int k) {
     if (k > text.length()) {
          throw std::invalid_argument(
               "Text is less than k in constructor! Invalid!");
     }
     order = k;
     size_t len = text.length();
     if (k == 0) {
          std::string emptyString = {};
          for (auto &a : text) {
               if (markovMaps[emptyString].find(a)
                    == markovMaps[emptyString].end()) {
                    markovMaps[emptyString][a] = 1;
                    alphabet.push_back(a);
               } else {
                    markovMaps[emptyString][a]++;
               }
          }
          std::sort(alphabet.begin(), alphabet.end());
          return;
     }
     for (size_t i = 0; i < len; i++) {
          if (alphabet.find(text[i]) == std::string::npos) {
               alphabet.push_back(text[i]);
          }
          std::string kgrams = i < len - (k - 1) ?
               text.substr(i, k) :
               text.substr(i, k).append(text.substr(0, i + k - len));
          char next = text[i + k - (i < len - k ? 0 : len)];
          if (markovMaps.find(kgrams) == markovMaps.end() ||
                    markovMaps[kgrams].find(next) == markovMaps[kgrams].end()) {
                    markovMaps[kgrams][next] = 1;
          } else {
                    markovMaps[kgrams][next]++;
          }
     }
     std::sort(alphabet.begin(), alphabet.end());
}
int MModel::freq(std::string kgram) const {
     if (kgram.length() != order) {
          throw std::invalid_argument(
               "kgram has incorrect length in freq! Invalid!");
     }
     if (markovMaps.find(kgram) == markovMaps.end()) {
          return 0;
     } else {
          unsigned int h = 0;
          for (const auto& a : markovMaps.at(kgram))
               h += a.second;
          return h;
     }
}
int MModel::freq(std::string kgram, char c) const {
     if (kgram.length() != order) {
          throw std::invalid_argument(
               "kgram has incorrect length in freq! Invalid!");
     }
     if (markovMaps.find(kgram) == markovMaps.end() ||
               markovMaps.at(kgram).find(c) == markovMaps.at(kgram).end()) {
               return 0;
     } else {
          return markovMaps.at(kgram).at(c);
     }
}
char MModel::kRand(std::string kgram) {
     if (kgram.length() != order) {
          throw std::invalid_argument(
               "kgram has incorrect length in kRand! Invalid!");
     }
     if (markovMaps.find(kgram) == markovMaps.end()) {
          throw std::runtime_error(
               "kgram string not found in any models in kRand! Invalid!");
     }
     int freqKgram = freq(kgram);
     int index = std::rand() % freqKgram;
     int i = 0;
     char random;
     for (auto &a : markovMaps[kgram]) {
          i += a.second;
          if (index < i) {
               random = a.first;
               break;
          }
     }
```

```cpp
    return random;
}
std::string MModel::generate(std::string kgram, unsigned int L) {
    if (kgram.length() != order) {
        throw std::invalid_argument(
            "kgram has incorrect length in generate! Invalid!");
    }
    if (L < order) {
        throw std::invalid_argument(
            "L in generate is too small! Invalid!");
    }
    if (markovMaps.find(kgram) == markovMaps.end()) {
        throw std::runtime_error(
            "kgram string not found in any models in generate! Invalid!");
    }
    std::string result{kgram};
    for (int i = 0; result.length() < L; i++) {
        std::string current = result.substr(i, order);  // Current kgram
        result.push_back(kRand(current));
    }
    return result;
}
std::ostream& operator<<(std::ostream& out, const MModel& m) {
    out << "The Order k = " << m.order << std::endl;
    out << "The Alphabet: ";
    for (size_t i = 0; i < m.alphabet.length(); i++) {
        if (i % 20 == 0) {
            out << std::endl << " ";
        }
        out << m.alphabet[i] << (i == m.alphabet.length() - 1 ?
            "" : ", ");
    }
    out << std::endl;
    out << "K-grams: " << std::endl;
    for (auto &k : m.markovMaps) {
        out << " \"" << k.first << "\": frequency = "
            << m.freq(k.first) << ", k+1-grams frequencies: "
            << std::endl;
        for (auto &a : k.second)
            out << " " << a.first << ": " << a.second << std::endl;
    }
    return out;
}
```

PS6: Kronos - Intro to Regular Expression

Assignment Description

This assignment involved using regular expressions to parse files of various Kronos InTouch time clock logs to analyze them, verifying the device's boot up timing, and noting whether or not these startups were fully successful or not. The purpose of this is so information can be gathered on what was occurring with the device at the time of the bootup failures, in order to eventually solve the problems occurring with the InTouch device. In order to help do this. PS6 asks us to scan the complete log files given and create a text file report chronologically describing each time the device was restarted, noting if it failed or succeeded in completely doing so, and giving the elapsed time for the sequence if the bootup was successful.

Key Algorithms, Data Structures, OO Design, Etc.

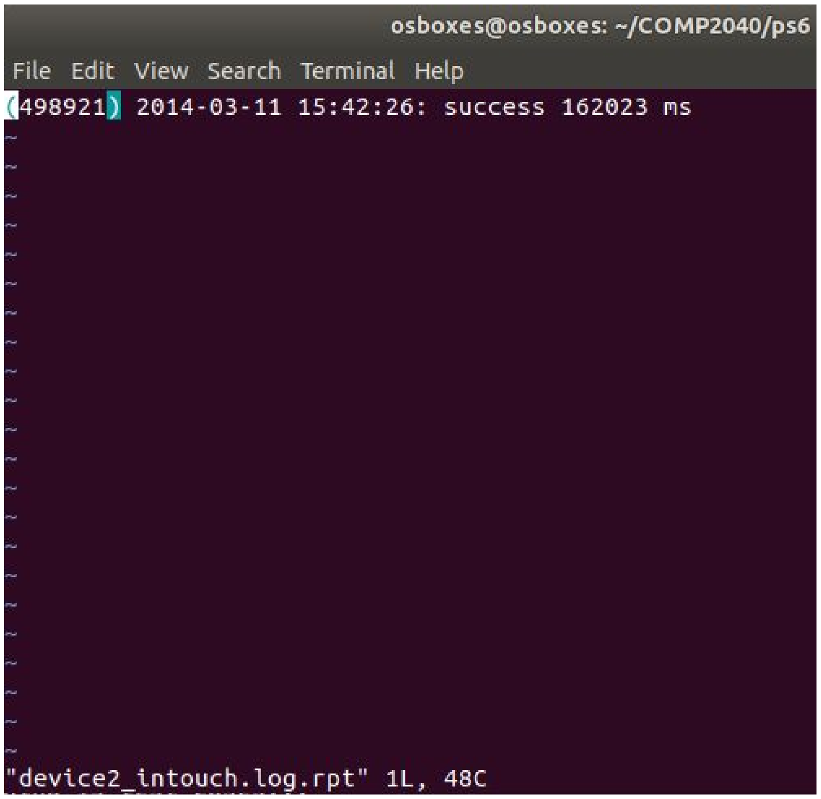
The usage of the Boost regex library is essential to the completion of this assignment, as it is used in order to find the specific pieces of data located within the given device log files, which are often tens of thousands of lines long. Four regular expressions were used in particular in order to find the specific lines of text that contain the needed information for the log report : one for the date, time, one for the boot sequence string indicating a startup is occurring, and a regex containing all three together.

On top of this, the Boost date and time functions were implemented in order to serve as a helping hand to computing the elapsed time of the successful startup sequences.

What I've Learned

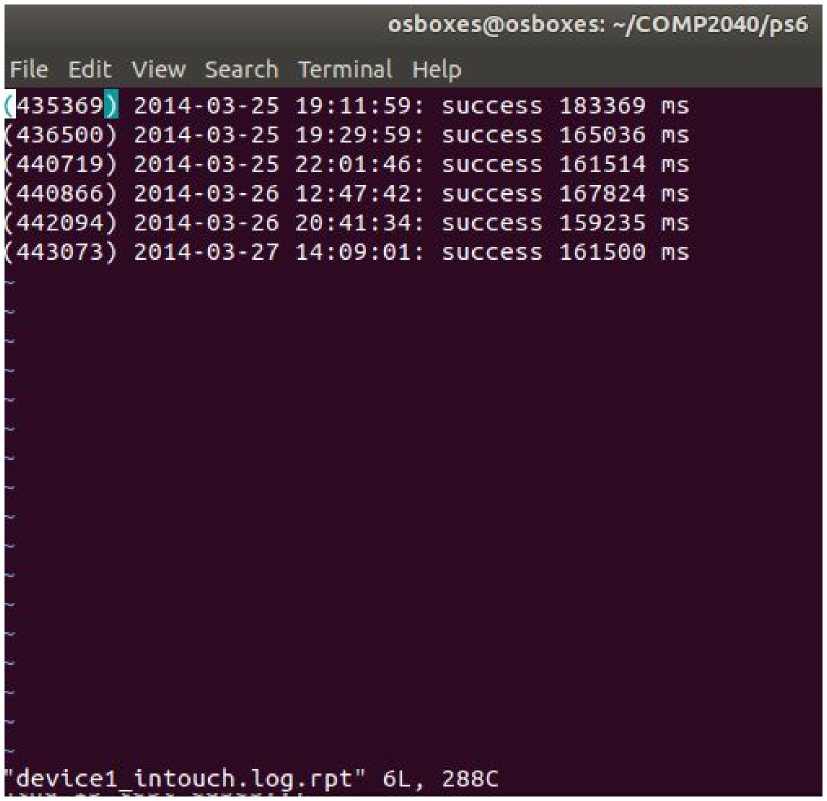
This assignment served as a solid introduction to regular expressions within the C++ language, and made me much more comfortable using the Boost regex library, as well as other regex libraries in general. The purpose of the assignment helped me better my proficiency in regards to outputting entire files, as well as provided an easy way to successfully parse a file. Lastly, the Boost date and time functions implemented were able to make my code more efficient in calculating the elapsed time based off of the string inputs I was receiving from the log file.

Screenshot(s) of Output:



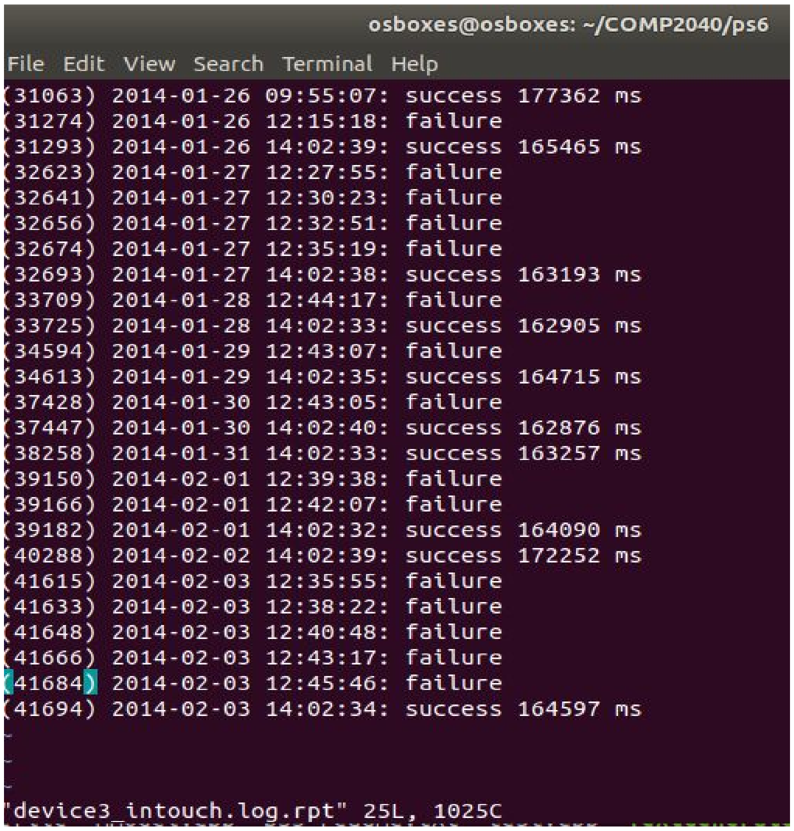
osboxes@osboxes: ~/COMP2040/ps6
File Edit View Search Terminal Help

```
(498921) 2014-03-11 15:42:26: success 162023 ms
~
"device2_intouch.log.rpt" 1L, 48C
```

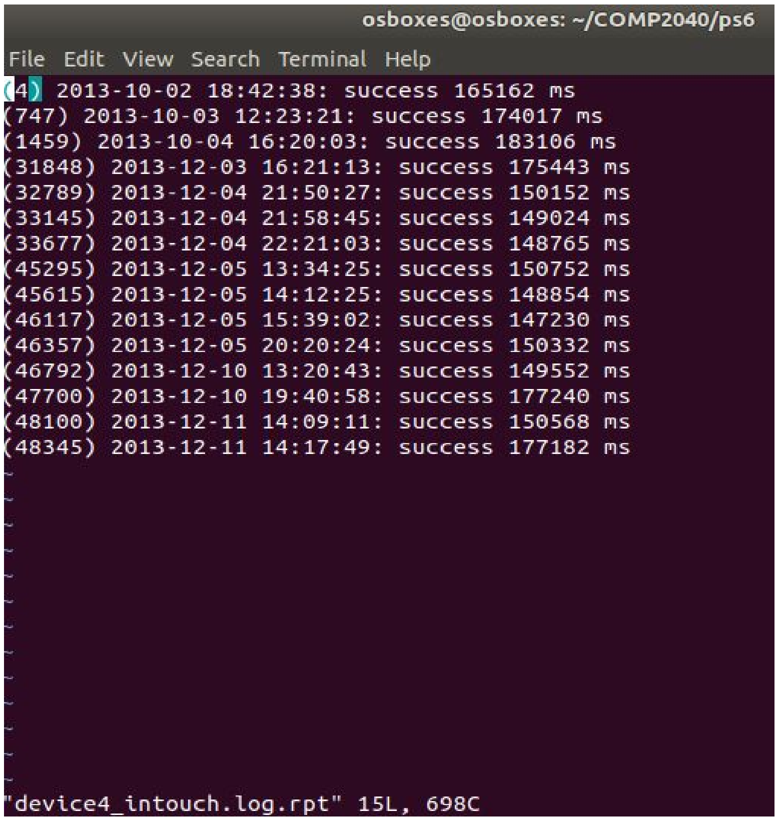


osboxes@osboxes: ~/COMP2040/ps6
File Edit View Search Terminal Help

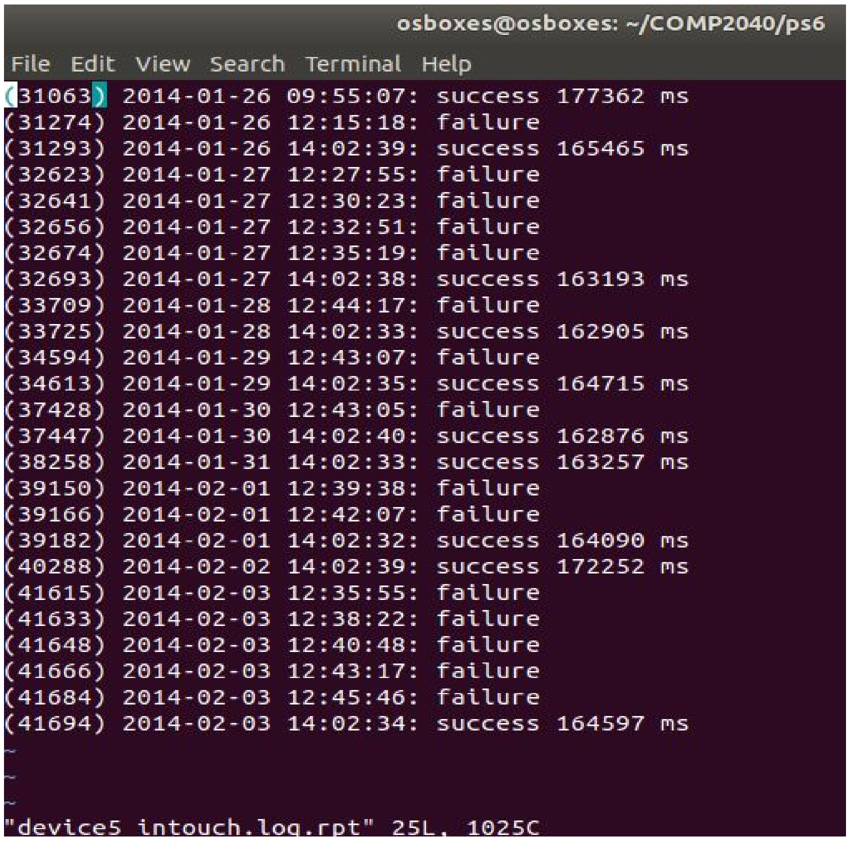
```
(435369) 2014-03-25 19:11:59: success 183369 ms
(436500) 2014-03-25 19:29:59: success 165036 ms
(440719) 2014-03-25 22:01:46: success 161514 ms
(440866) 2014-03-26 12:47:42: success 167824 ms
(442094) 2014-03-26 20:41:34: success 159235 ms
(443073) 2014-03-27 14:09:01: success 161500 ms
~
"device1_intouch.log.rpt" 6L, 288C
```



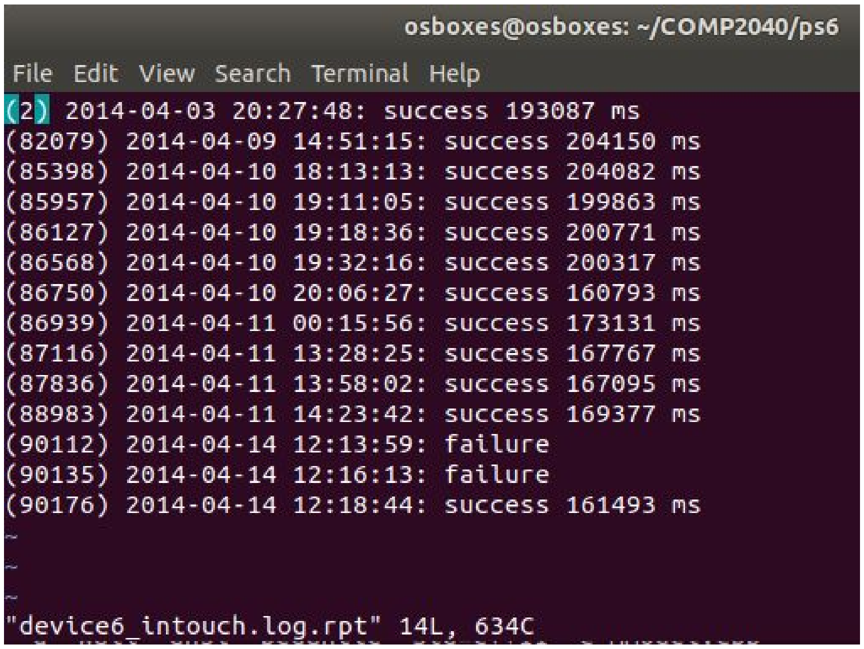
```
osboxes@osboxes: ~/COMP2040/ps6
File  Edit  View  Search  Terminal  Help
(31063) 2014-01-26 09:55:07: success 177362 ms
(31274) 2014-01-26 12:15:18: failure
(31293) 2014-01-26 14:02:39: success 165465 ms
(32623) 2014-01-27 12:27:55: failure
(32641) 2014-01-27 12:30:23: failure
(32656) 2014-01-27 12:32:51: failure
(32674) 2014-01-27 12:35:19: failure
(32693) 2014-01-27 14:02:38: success 163193 ms
(33709) 2014-01-28 12:44:17: failure
(33725) 2014-01-28 14:02:33: success 162905 ms
(34594) 2014-01-29 12:43:07: failure
(34613) 2014-01-29 14:02:35: success 164715 ms
(37428) 2014-01-30 12:43:05: failure
(37447) 2014-01-30 14:02:40: success 162876 ms
(38258) 2014-01-31 14:02:33: success 163257 ms
(39150) 2014-02-01 12:39:38: failure
(39166) 2014-02-01 12:42:07: failure
(39182) 2014-02-01 14:02:32: success 164090 ms
(40288) 2014-02-02 14:02:39: success 172252 ms
(41615) 2014-02-03 12:35:55: failure
(41633) 2014-02-03 12:38:22: failure
(41648) 2014-02-03 12:40:48: failure
(41666) 2014-02-03 12:43:17: failure
(41684) 2014-02-03 12:45:46: failure
(41694) 2014-02-03 14:02:34: success 164597 ms
~
~
"device3_intouch.log.rpt" 25L, 1025C
```



```
osboxes@osboxes: ~/COMP2040/ps6
File  Edit  View  Search  Terminal  Help
(4) 2013-10-02 18:42:38: success 165162 ms
(747) 2013-10-03 12:23:21: success 174017 ms
(1459) 2013-10-04 16:20:03: success 183106 ms
(31848) 2013-12-03 16:21:13: success 175443 ms
(32789) 2013-12-04 21:50:27: success 150152 ms
(33145) 2013-12-04 21:58:45: success 149024 ms
(33677) 2013-12-04 22:21:03: success 148765 ms
(45295) 2013-12-05 13:34:25: success 150752 ms
(45615) 2013-12-05 14:12:25: success 148854 ms
(46117) 2013-12-05 15:39:02: success 147230 ms
(46357) 2013-12-05 20:20:24: success 150332 ms
(46792) 2013-12-10 13:20:43: success 149552 ms
(47700) 2013-12-10 19:40:58: success 177240 ms
(48100) 2013-12-11 14:09:11: success 150568 ms
(48345) 2013-12-11 14:17:49: success 177182 ms
~
~
~
~
~
~
~
~
~
~
~
~
~
"device4_intouch.log.rpt" 15L, 698C
```


osboxes@osboxes: ~/COMP2040/ps6
File Edit View Search Terminal Help

```
(31063) 2014-01-26 09:55:07: success 177362 ms
(31274) 2014-01-26 12:15:18: failure
(31293) 2014-01-26 14:02:39: success 165465 ms
(32623) 2014-01-27 12:27:55: failure
(32641) 2014-01-27 12:30:23: failure
(32656) 2014-01-27 12:32:51: failure
(32674) 2014-01-27 12:35:19: failure
(32693) 2014-01-27 14:02:38: success 163193 ms
(33709) 2014-01-28 12:44:17: failure
(33725) 2014-01-28 14:02:33: success 162905 ms
(34594) 2014-01-29 12:43:07: failure
(34613) 2014-01-29 14:02:35: success 164715 ms
(37428) 2014-01-30 12:43:05: failure
(37447) 2014-01-30 14:02:40: success 162876 ms
(38258) 2014-01-31 14:02:33: success 163257 ms
(39150) 2014-02-01 12:39:38: failure
(39166) 2014-02-01 12:42:07: failure
(39182) 2014-02-01 14:02:32: success 164090 ms
(40288) 2014-02-02 14:02:39: success 172252 ms
(41615) 2014-02-03 12:35:55: failure
(41633) 2014-02-03 12:38:22: failure
(41648) 2014-02-03 12:40:48: failure
(41666) 2014-02-03 12:43:17: failure
(41684) 2014-02-03 12:45:46: failure
(41694) 2014-02-03 14:02:34: success 164597 ms
~
~
~
"device5_intouch.log.rpt" 25L, 1025C
```


osboxes@osboxes: ~/COMP2040/ps6
File Edit View Search Terminal Help

```
(2) 2014-04-03 20:27:48: success 193087 ms
(82079) 2014-04-09 14:51:15: success 204150 ms
(85398) 2014-04-10 18:13:13: success 204082 ms
(85957) 2014-04-10 19:11:05: success 199863 ms
(86127) 2014-04-10 19:18:36: success 200771 ms
(86568) 2014-04-10 19:32:16: success 200317 ms
(86750) 2014-04-10 20:06:27: success 160793 ms
(86939) 2014-04-11 00:15:56: success 173131 ms
(87116) 2014-04-11 13:28:25: success 167767 ms
(87836) 2014-04-11 13:58:02: success 167095 ms
(88983) 2014-04-11 14:23:42: success 169377 ms
(90112) 2014-04-14 12:13:59: failure
(90135) 2014-04-14 12:16:13: failure
(90176) 2014-04-14 12:18:44: success 161493 ms
~
~
~
"device6_intouch.log.rpt" 14L, 634C
```

Source Code:

Makefile:

```
CC=g++
OBJ= stdin_boost.o
CFLAGS=-g -Wall -ansi -pedantic -std=c++14
LBOOSTFLAGS=-lboost_regex -lboost_date_time
all: $(OBJ)
        $(CC) $(CFLAGS) -o ps6 stdin_boost.o $(LBOOSTFLAGS)
ps6: stdin_boost.o
        $(CC) $(CFLAGS) -o ps6 stdin_boost.o $(LBOOSTFLAGS)
stdin_boost.o: stdin_boost.cpp
        $(CC) $(CFLAGS) -c stdin_boost.cpp
clean:
        -@rm -rf *.o 2>/dev/null || true
```

Stdin_boost.cpp:

```
// Copyright Jacob Leboeuf 2020
#include <boost/regex.hpp>
#include <string>
#include <iostream>
#include <fstream>
#include "boost/date_time/gregorian/gregorian.hpp"
#include "boost/date_time/posix_time/posix_time.hpp"
using namespace boost::posix_time; //NOLINT
int main(int argc, char* argv[]) {
  std::ofstream output;
  std::ifstream input(argv[1]);
  std::string line;
  std::string rDate = "^[0-9]{4}-[0-9]{2}-[0-9]{2} ";
  std::string rTime = "[0-9]{2}:[0-9]{2}:[0-9]{2}";
  std::string sBoot = "(.*)log.c.166(.*)";
  std::string sSuccess =
      "(.*)oejs.AbstractConnector:Started SelectChannelConnector(.*)";
  if (!input.is_open()) {
    std::cout << "Cannot open file!" << std::endl;
    return 1;
  }
  std::string reportName(std::string(argv[1]) + ".rpt");
  output.open(reportName.c_str());
  boost::regex boot(rDate + rTime + sBoot);
  boost::regex succ(rDate + rTime + sSuccess);
  if (input.is_open()) {
    bool isFinished = true;
    int lineCount = 0;
    ptime b;
    while (getline(input, line)) {
      lineCount++;
      if (boost::regex_match(line, succ)) {
        isFinished = true;
        ptime e(time_from_string(line.substr(0, 24)));
        time_duration t = e - b;
        output << "success " << t.total_milliseconds() << " ms"
               << std::endl;
      } else if (boost::regex_match(line, boot)) {
          if (isFinished == false) {
            output << "failure" << std::endl;
          }
          output << "(" << lineCount << ") ";
          output << line.substr(0, 21);
          ptime b2(time_from_string(line.substr(0, 20)));
          b = b2;
          isFinished = false;
        }
    }
    output.close();
    input.close();
  }
  return 0;
}
```