Chris Olah
COMP IV Sec 202: Project Portfolio
Spring 2022

<u>Contents:</u>

PS0 Hello World with SFML

The Assignment

The main goal of PS0 was to set up our Linux build environment and to test out the SFML audio/graphics library. The assignment required a sprite to be displayed on the SFML window and have the ability to move. We had to make sure it moved and responded to keystrokes and include an additional feature.

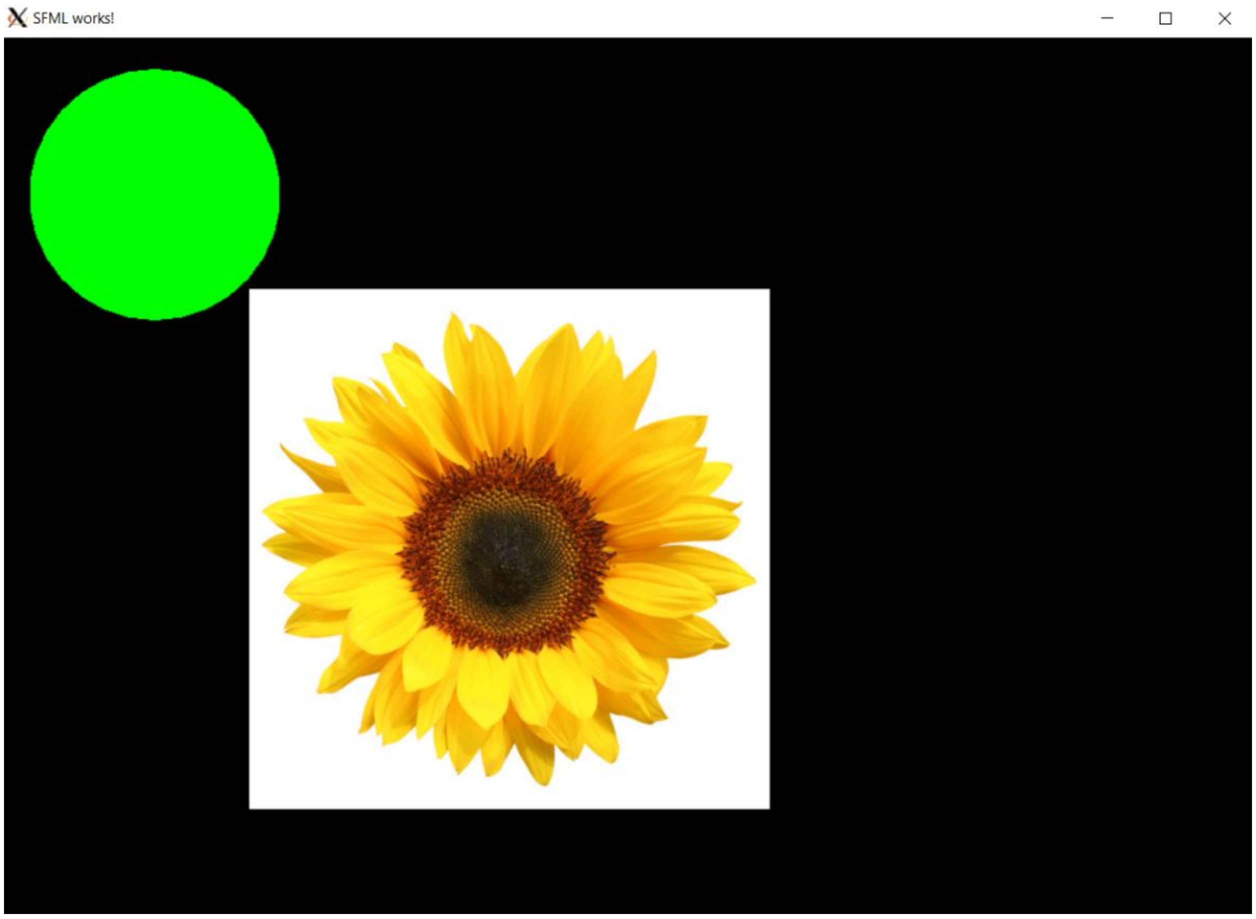
Key Concepts

I used basic loops for keeping the window open and for moving the sprites around. Then, I created a program that uses SFML's sprite class to display a moving image on an SFML window and added some basic controls to move it around once certain keys on the keyboard are pressed. Additionally, I used window.setFrameLimit() method to make sure the sprite didn't move too fast.

What I Learned

I learned to work with SFML and familiarized myself with it's environment and features. That includes drawing an image sprite and moving it around in an SFML window. I was also able to learn more about virtual machines and its benefits. Furthermore, I strengthened my knowledge of using command line and getting used to the Linux environment.

Output/Screenshot



PS0 Source Code:

main.cpp

```
/**************************************************
Computing IV Project 0 - Setting Up Coding Environemnt with SFML
Author: Christopher Olah
Date: January 24, 2022
Code Extended From: https://www.sfml-dev.org/documentation/2.5.1/
 **************************************************/

#include

int main()
{
        // Create the main window
        sf::RenderWindow window(sf::VideoMode(1000, 700), "SFML works!");

        // Load and display sprite
        sf::Texture texture;
        if (!texture.loadFromFile("sprite.jpg"))
                        return EXIT_FAILURE;
        sf::Sprite sprite(texture);

        // Move Sprite
        int changeX = 200;
        int changeY = 200;
        sprite.move(changeX, changeY);

        // Display Green Circle
        sf::CircleShape shape(100.f);
        shape.setFillColor(sf::Color::Green);

        // Move Green Circle
        shape.move(25, 25);

        // Start Window Display Loop
        while (window.isOpen())
        {
                // Press Up Key to Move Sprite Upwards
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
                        sprite.move(changeX, changeY - 2);
                }

                // Press Down Key to Move Sprite Downwards
                else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
                        sprite.move(changeX, changeY + 2);
                }

                // Press R Key to Change Sprite Color to Green
                else if (sf::Keyboard::isKeyPressed(sf::Keyboard::R)) {
                        sprite.setColor(sf::Color::Green);
                }
                else {
                        changeX = 0;
                }

                //Press Left key to Move Sprite Leftwards
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
                        sprite.move(changeX - 2, changeY);
                }

                // Press Right Key to Move Sprite Rightwards
                else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
                        sprite.move(changeX + 2, changeY);
                }
                else {
                        changeY = 0;
                }

                // Top Wall Collision
                if (sprite.getPosition().y < 0)
                        sprite.setPosition(sprite.getPosition().x, 0);

                // Bottom Collision
                if (sprite.getPosition().y + sprite.getGlobalBounds().height > 700)
                        sprite.setPosition(sprite.getPosition().x, 700 - sprite.getGlobalBounds().height);

                // Left Wall Collision
                if (sprite.getPosition().x < 0)
                        sprite.setPosition(0, sprite.getPosition().y);

                // Right Wall Collision
                if (sprite.getPosition().x + sprite.getGlobalBounds().width > 1000)
                        sprite.setPosition(1000 - sprite.getGlobalBounds().width, sprite.getPosition().y);

                // Process Events
                sf::Event event;
                while (window.pollEvent(event))
                {

                        if (event.type == sf::Event::Closed)
                        window.close();
                }

                // Clear Screen
                window.clear();

                // Display sprite
                window.draw(sprite);

                // Display Green Circle
                window.draw(shape);

                // Update Window
```

```
            window.display();
        }

        return 0;
}
```

PS1a Linear Feedback Shift Register and Unit Testing

The Assignment

The goal of PS1a was to write a data type that simulates the operation of a linear feedback shift register and then uses it to implement a simple form of encryption for digital pictures. I applied XOR to create a register that shifts a bit one position to the left and replaces the new spot with the XOR value of the bit shifted and the bit at a given tap position in the register. I had to implement the Fibonacci LFSR for this program.

Key Concepts

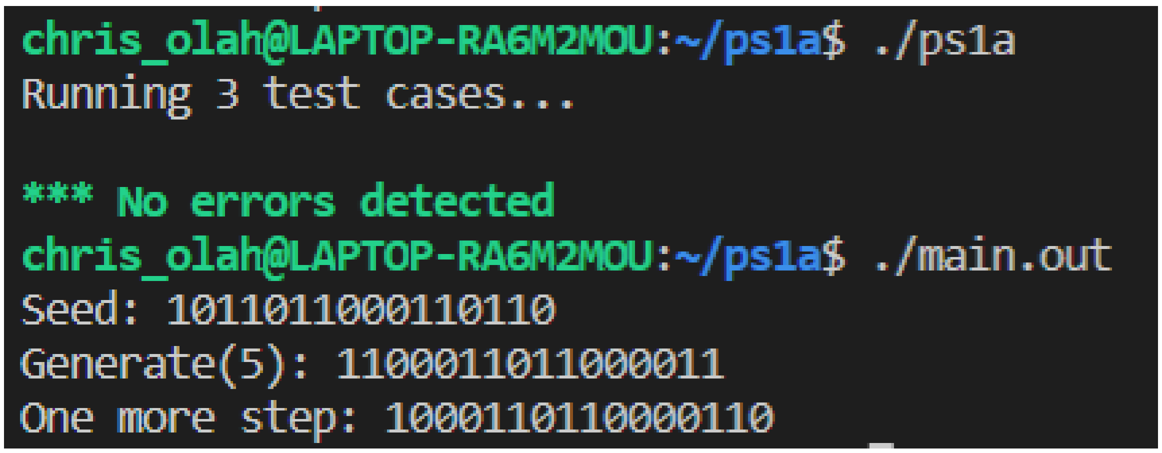
The representation I used for the register bits was strings. I used this because strings are easy to work with. You can use the string operations to get certain bits and to get the seed then just shift the string to the left with a loop. Additionally, another key function in this program was the function generate as it returned a k-bit integer by simulating k steps of the LFSR. It was accomplished by initializing a variable to zero and for each bit that is extracted you double the variable and add the bit returned by the function step(). The use of XOR and calculating the newBit within the step() function is listed below:

```
int newBit = (((tapPosition1 ^ tapPosition2) ^ tapPosition3) ^ tapPosition4);
```

What I Learned

Although the concept of xor-ing values and shifting them around seemed easy enough to comprehend, it was a little challenging to implement it in the form of taps. This assignment taught me a lot about testing in C++. Using the Boost test framework made it more efficient to run and check for errors and exceptions. Although it was challenging implementing for the first time, I was able to understand its purpose and succeed in testing my code.

Output/Screenshot



```
chris_olah@LAPTOP-RA6M2MOU:~/ps1a$ ./ps1a
Running 3 test cases...

*** No errors detected
chris_olah@LAPTOP-RA6M2MOU:~/ps1a$ ./main.out
Seed: 1011011000110110
Generate(5): 1100011011000011
One more step: 1000110110000110
```

PS1a Source Code:

Makefile

```
CC = g++
CFLAGS = -Wall -Werror -pedantic -std=c++14
Boost = -lboost_unit_test_framework

all:
	make ps1a main.out

ps1a: test.o FibLFSR.o
	$(CC) $(CFLAGS) -o ps1a test.o FibLFSR.o $(Boost)

main.out: main.o FibLFSR.o
	$(CC) main.o FibLFSR.o -o main.out

main.o: main.cpp FibLFSR.h
	$(CC) $(CFLAGS) -c main.cpp

test.o: test.cpp
	$(CC) $(CFLAGS) -c test.cpp

FibLFSR.o: FibLFSR.h FibLFSR.cpp main.cpp
	$(CC) $(CFLAGS) -c FibLFSR.cpp main.cpp

clean:
	rm -f *.o *~ ps1a
```

main.cpp

```
/**********************************************************
Author: Chris Olah
Course: Computing IV (COMP.2040)
Assignment: Project 1A
Professor: Dr. Daly
Date: February 1 2022
**********************************************************/

#include <iostream>
#include "FibLFSR.h"

// Including main.cpp source file for Makefile
int main(int argc, char* argv[]) {
			FibLFSR flfsr("1011011000110110");
    std::cout << "Seed: " << flfsr << std::endl;

    flfsr.generate(5);
    std::cout << "Generate(5): " << flfsr << std::endl;

    flfsr.step();
    std::cout << "One more step: " << flfsr << std::endl;

    return 0;
}
```

FibLFSR.h

```
/************************************************************
Author: Chris Olah
Course: Computing IV (COMP.2040)
Assignment: Project 1A
Professor: Dr. Daly
Date: February 1 2022
************************************************************/

#pragma once

#include <iostream>
#include <string>
#include <vector>

using std::string;
using std::ostream;

// Class structure for FibLFSR object
class FibLFSR {
	public:
		// Constructor to create LFSR with initial seed
		FibLFSR(string seed);

		// Simulate one step and return new bit as 0 or 1
		int step();

		 // Simulate k steps and return k-bit integer
		 int generate(int k);

		 // Overloading the << operator to display its current register value in printable form
friend ostream& operator<< (ostream& os, const FibLFSR& flfsr);

private:
// String to store seed
string _seed;

// Seed tap positions
int _tap1 = 0;
int _tap2 = 2;
int _tap3 = 3;
int _tap4 = 5;
}
```

FibLFSR.cpp

```
/***********************************************************
Author: Chris Olah
Course: Computing IV (COMP.2040)
Assignment: Project 1A
Professor: Dr. Daly
Date: February 1 2022
***********************************************************/

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

// FibLFSR constructor definition for initial seed
FibLFSR::FibLFSR(string seed) {
        _seed = seed;
}

// FibLFSR step function definition
int FibLFSR::step() {
    char tapPosition1 = _seed[_tap1];
    char tapPosition2 = _seed[_tap2];
    char tapPosition3 = _seed[_tap3];
    char tapPosition4 = _seed[_tap4];

    // Calculated new bit given the tap positions
    int newBit = (((tapPosition1 ^ tapPosition2) ^ tapPosition3) ^ tapPosition4);

    int seedLength = _seed.size();

    // Shift bits to left
    for (int i = 0; i < (seedLength) - 1; i++) {
        _seed[i] = _seed[i + 1];
    }

     // Assign last bit to end of seed (after shift to left)
     _seed.back() = newBit;

     // Return new bit value
     return _seed.back();
}
```

```

// FibLFSR generate function definition
int FibLFSR::generate(int k) {
    int extractedBit = 0;

    for(int i = 0; i < k; i++) {
        extractedBit = extractedBit * 2 + step();
    }

    // return extracted generated bit after k steps
    return extractedBit;
}

// FibLFSR overload insertion operator definition
ostream& operator<<(ostream& os, const FibLFSR& flfsr) {
     //output seed string to os
     os << flfsr._seed;

     return os;
}

```

test.cpp

```
/*********************************************************
Author: Chris Olah
Course: Computing IV (COMP.2040)
Assignment: Project 1A
Professor: Dr. Daly
Date: February 1 2022
CODE EXTENDED FROM Dr. Rykalova (Updated 1/31/2020)
*********************************************************/
#include <iostream>
#include <string>
#include <sstream>

#include "FibLFSR.h"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include

// Given test cases for testing 16 bit seed
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);
}

// Tests 10 bit seed with all 0’s
BOOST_AUTO_TEST_CASE(twentyBitsZeroes) {

    FibLFSR l("00000000000000000000");

    BOOST_REQUIRE(l.step() == 0);
    BOOST_REQUIRE(l.step() == 0);
    BOOST_REQUIRE(l.step() == 0);
    BOOST_REQUIRE(l.step() == 0);
    BOOST_REQUIRE(l.step() == 0);
    BOOST_REQUIRE(l.step() == 0);
    BOOST_REQUIRE(l.step() == 0);
    BOOST_REQUIRE(l.step() == 0);
    BOOST_REQUIRE(l.step() == 0);
    BOOST_REQUIRE(l.step() == 0);
    BOOST_REQUIRE(l.step() == 0);

    FibLFSR l2("00000000000000000000");
    BOOST_REQUIRE(l2.generate(9) == 0);

}

// Tests to make sure constructor stores seed correctly
// Also tests overloading << insertion operator
BOOST_AUTO_TEST_CASE(fiveBitsTapAtZero) {
    FibLFSR l("00000001111111111001010101011");
    std::stringstream buffer;
    buffer << l;

    BOOST_REQUIRE(buffer.str().compare("00000001111111111001010101011") == 0);

    FibLFSR l2("011011");
    buffer.str("");
    buffer.clear();
    buffer << l2;

    BOOST_REQUIRE(buffer.str().compare("011011") == 0);
}

```

PS1b Using LFSR to Encode and Decode Images

The Assignment

For PS1b we had to write a program that read three arguments from the command line including the source image filename, output image filename and FibLFSR seed. We had to use SFML to load the source image and display it on its window. Then we had to use our debugged FibLFSR class to encode or decode the image and display the output to the window. I built on the previous assignment and wrote PhotoMagic.cpp to carry out assignment. We obtained our desired outputs by performing XOR to the color components (red, blue, green). When I run the code from command line, it will show the input file or the source file and the encrypted file. I used SFML for displaying the image.

Key Concepts

One of the most significant aspects of this code was carried out in the previous part in the form of creating the linear shift register. However, this part focuses on encoding/decoding the images which was accomplished by accessing each pixel for each RGB value and XOR-ing it to get random color values.

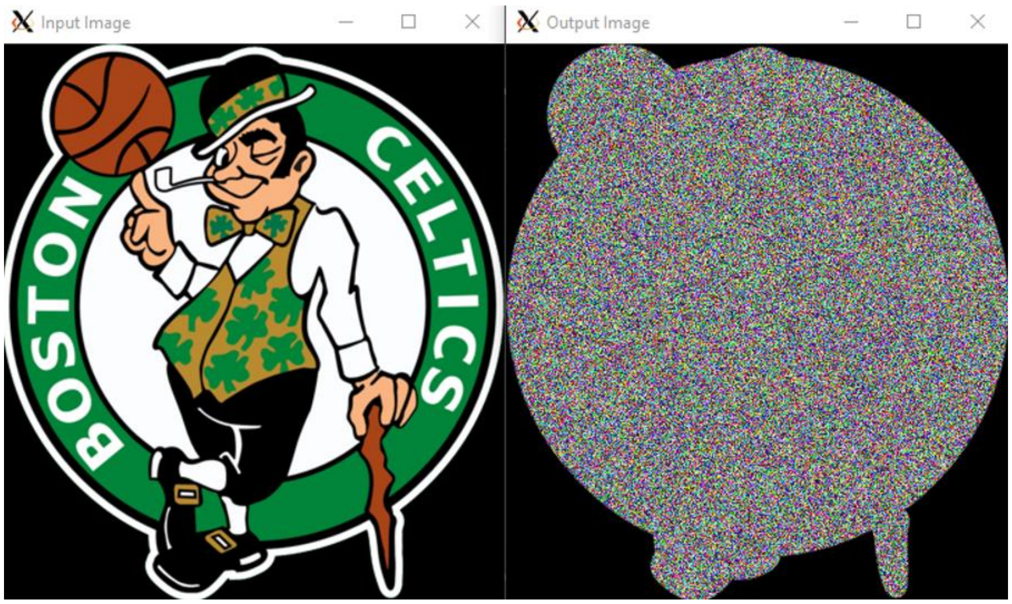
```
        pixel = inputImage.getPixel(x, y);
        pixel.r = (pixel.r ^ flfsr->generate(8));
        pixel.g = (pixel.g ^ flfsr->generate(8));
        pixel.b = (pixel.b ^ flfsr->generate(8));
        inputImage.setPixel(x, y, pixel);
```

What I Learned

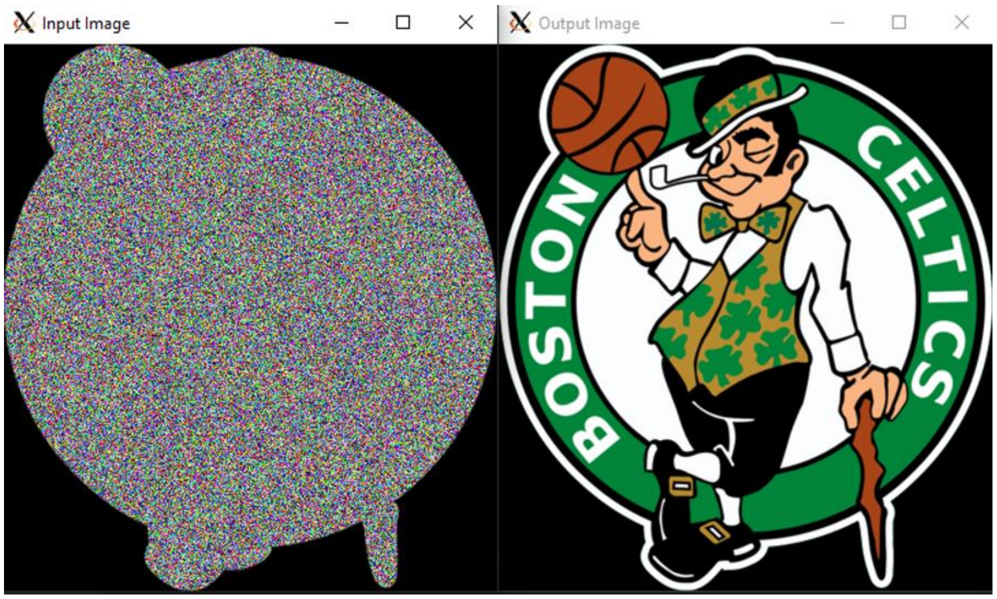
In this assignment I learned more about working with the SFML environment and using sf::Image. I learned more about accessing images and altering its properties. Additionally, I came across many small errors in the code that strengthened my critical thinking skills.

Output/Screenshot

Encode



Decode



PS1b Source Code:

Makefile

```
CC = g++
CFLAGS = -Wall -Werror -pedantic -std=c++14
LIBS = -lsfml-graphics -lsfml-window -lsfml-system

all:
	make PhotoMagic

PhotoMagic: PhotoMagic.o FibLFSR.o
	$(CC) $(CFLAGS) PhotoMagic.o FibLFSR.o -o PhotoMagic $(LIBS)

PhotoMagic.o: PhotoMagic.cpp
	$(CC) $(CFLAGS) -c PhotoMagic.cpp

FibLFSR.o: FibLFSR.cpp FibLFSR.h
	$(CC) $(CFLAGS) -c FibLFSR.cpp

clean:
	rm -f *.o *~ PhotoMagic
```

PhotoMagic.cpp

```
/************************************************************
* @file PhotoMagic.cpp
* @author: Chris Olah
* @copyright Umass Lowell February 7 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 1B
* Professor: Dr. Daly
************************************************************/

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

/**
* @brief Tests if this string starts with the specified prefix.
* @param inputImage Original image to be decrypted into output Image
* @param flfsr Binary seed of FibLFSR class for encryption/decryption pro
cess
* @since 1.0
*/
void transform(sf::Image& inputImage, FibLFSR* flfsr) {
	sf::Vector2u imageSize = inputImage.getSize(); ///< Size of input image (width * height)

	sf::Color pixel; ///< Single image pixel

	for (int x = 0; x < (int) imageSize.x; x++) { //< Go through every pixel and encrypt/decrypt them
		for (int y = 0; y < (int) imageSize.y; y++) {
			pixel = inputImage.getPixel(x, y);
			pixel.r = (pixel.r ^ flfsr->generate(8));
			pixel.g = (pixel.g ^ flfsr->generate(8));
			pixel.b = (pixel.b ^ flfsr->generate(8));
			inputImage.setPixel(x, y, pixel);
		}
	}
}

```

```
/**
* @brief Dislay original input image along with encrypted/decrypted output image
* @param inputImage Original image to be decrypted into output Image
* @param flfsr Binary seed of FibLFSR class for encryption/decryption process
* @since 1.0
*/
void draw(sf::Image& inputImage, FibLFSR* flfsr) {
    sf::Vector2u imageSize = inputImage.getSize(); //< Size of input image (width * height)

    sf::Image outputImage = inputImage; //< Create output image and copy input image to it

    transform(outputImage, flfsr); //< Encrypt/Decrypt given output image

    sf:: Texture inputTexture, outputTexture;
    inputTexture.loadFromImage(inputImage);
    outputTexture.loadFromImage(outputImage);

    sf::Sprite inputSprite, outputSprite;
    inputSprite.setTexture(inputTexture);
    outputSprite.setTexture(outputTexture);

    sf::RenderWindow inputWindow(sf::VideoMode(imageSize.x, imageSize.y), "Input      Image");
sf::RenderWindow outputWindow(sf::VideoMode(imageSize.x, imageSize.y), "Output Image");

    while (inputWindow.isOpen() && outputWindow.isOpen()) { //< Display input and output image in two seperate windows
        sf::Event event;
        while (inputWindow.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                inputWindow.close();
        }
        while (outputWindow.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
            outputWindow.close();
        }

        inputWindow.clear(sf::Color::Black);
        inputWindow.draw(inputSprite); //< Draw input image
        inputWindow.display();

        outputWindow.clear(sf::Color::Black);
        outputWindow.draw(outputSprite); //< Draw output image
        outputWindow.display();
    }

    inputImage = outputImage; //< Maintain original image
}

int main(int argc, char* argv[]) {
    sf::Image inputImage; //< Create image input image

    string inputFileName(argv[1]); //< Take second command store as input file name
    string outputFileName(argv[2]); //< Take third command store as output file name
    string seed = argv[3]; //< Take fourth command and store it as seed

    inputImage.loadFromFile(inputFileName); //< Store input file image into inputImage

    FibLFSR flfsr(seed); //< Store seed value into FibLFSR

    draw(inputImage, &flfsr); //< Display input and encrypted/decrypted output image seed

    if (!inputImage.saveToFile(outputFileName)) //< Save encrypted/decrypted image
        return -1;

    return 0;
}
```

FibLFSR.h

```
/***********************************************************
* @file FibLFSR.h
* @author: Chris Olah
* @copyright Umass Lowell February 7 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 1B
* Professor: Dr. Daly
***********************************************************/
#pragma once

#include <iostream>
#include <string>
#include <vector>

using std::string;
using std::ostream;

/**
* @brief Class to describe FibLFSR object
* @since 1.0
*/
class FibLFSR { //< Class structure for FibLFSR object
        public:
              FibLFSR(string seed); //< Constructor to create LFSR with initial seed


             int step(); //< Simulate one step and return new bit as 0 or 1


             int generate(int k); //< Simulate k steps and return k-bit integer


             friend ostream& operator<< (ostream& os, const FibLFSR& flfsr);

       private:
             string _seed; //< String to store seed


             int _tap1 = 0;
```

```
    int _tap2 = 2;
    int _tap3 = 3;
    int _tap4 = 5; //< Seed tap positions
};
```

FibLFSR.cpp

```
/*************************************************************
* @file FibLFSR.cpp
* @author: Chris Olah
* @copyright Umass Lowell February 7 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 1B
* Professor: Dr. Daly
*************************************************************/

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

/**
* @brief FibLFSR constructor definition for initial seed
* @param seed value of seed
* @since 1.0
*/
FibLFSR::FibLFSR(string seed) { //< FibLFSR constructor definition for initial seed
	_seed = seed;
}

/**
* @brief Simulate one step and calculate new extracted bit
* @return Extracted bit calculated from xor of tap positions (0 or 1)
* @since 1.0
*/
int FibLFSR::step() { //< FibLFSR step function definition
	char tapPosition1 = _seed[_tap1];
	char tapPosition2 = _seed[_tap2];
	char tapPosition3 = _seed[_tap3];
	char tapPosition4 = _seed[_tap4];

	int newBit = (((tapPosition1 ^ tapPosition2) ^ tapPosition3) ^ tapPosition4);

	int seedLength = _seed.size(); //< Get seed length

	for (int i = 0; i < (seedLength) - 1; i++) { //< Shift bits to left
		_seed[i] = _seed[i + 1];
```

```
    }

    _seed.back() = newBit; //< Assign last bit to end of seed (after shift to left)

    return _seed.back(); //< Return new bit value
}

/**
* @brief FibLFSR constructor definition for initial seed
* @param k value of how many steps (step()) to complete
* @return k-bit integer after k steps generated
* @since 1.0
*/
int FibLFSR::generate(int k) { //< FibLFSR generate function definition
    int extractedBit = 0; //< Declare/initialize extract bit

    for(int i = 0; i < k; i++) {
        extractedBit = extractedBit * 2 + step();
    }

    return extractedBit;
}

/**
* @brief Overloading the << operator to display its current register valu
e in printable form
* @param os ostream to output LFSR
* @param flfsr pass flfsr object to output
* @return output of current register value
* @since 1.0
*/
ostream& operator<<(ostream& os, const FibLFSR& flfsr) { //< FibLFSR overload operator
    os << flfsr._seed;

    return os;
}

```

PS2a Loading Universe Files; Body Class; Graphics

The Assignment

The assignment was to load and display a static universe. We were provided with images and input files that we used to create the universe. The program took in inputs from a file using cin to read the file. I overloaded the >> operator as instructed. The text file consisted of the number of particles, radius of the universe, x and y co-ordinates of the initial position, x and y components of the initial velocity and mass. These values were followed by the name of the file used to display that particular planet. We had to use smart pointers for the purpose of this program.

Key Concepts

One of the important aspects to this program was reading in the input values from the text file as they contained all the required data for our program to run efficiently.

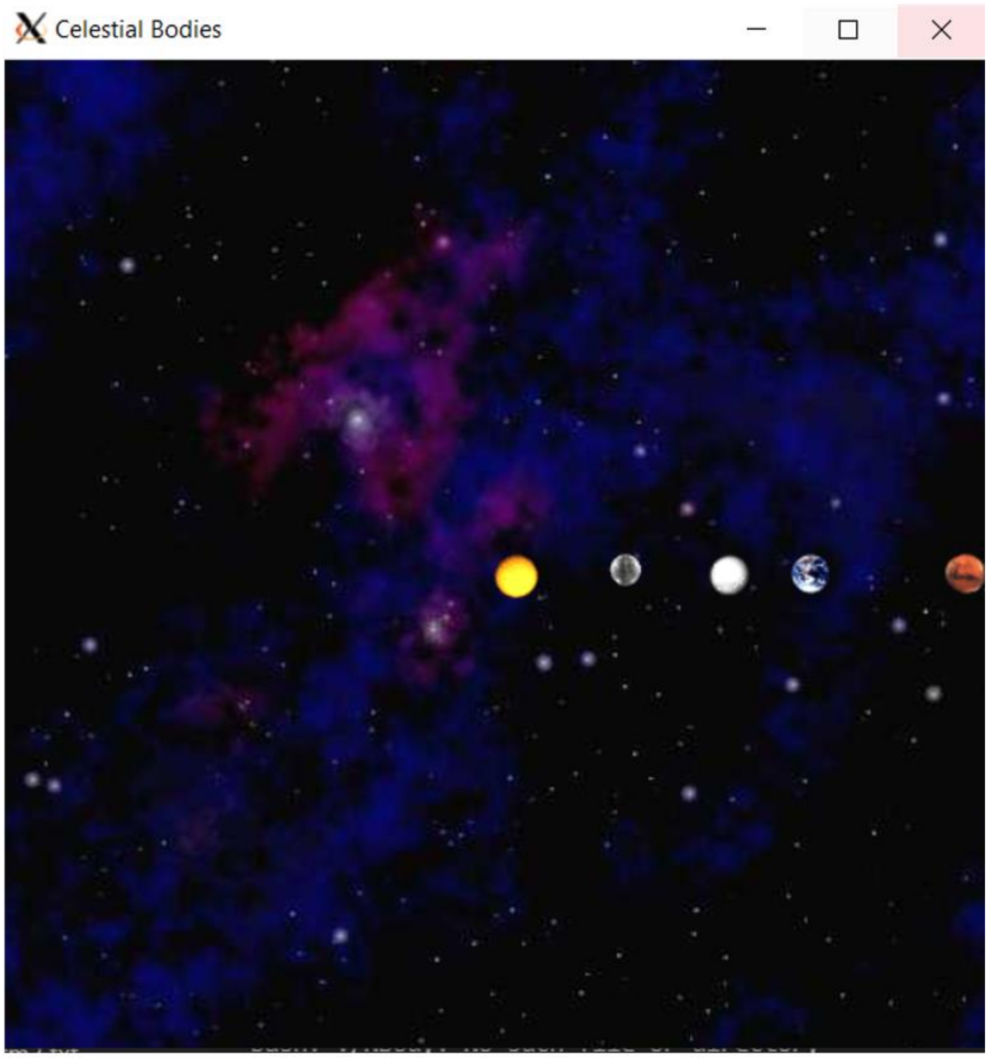
```
shared_ptrplanet(new CelestialBody);
cin >> *planet;
planet->radius(radiusofUni);
planet->position();
universe.body.push_back(planet);
```

Additionally, the goal of the assignment was to have the planets appear at their initial positions which was obtained by using simple functions to calculate its position using the x and y components provided to us in the input files.

What I Learned

I learned how to convert the coordinates from planetary units to SFML's units. I also learned more about smart pointers in the process of applying it to this program and reading in values from text files correctly, and this was a very new concept for me so I'm glad I was able to understand smart pointers because they seem very essential for programming.

Output/Screenshot



PS1b Source Code:

Makefile

```
CC= g++ -std=c++17
CFLAGS= -lsfml-graphics -lsfml-window -lsfml-system
OBJECTS = main.o CelestialBody.o

all:
	make NBody

NBody :$(OBJECTS)
	$(CC) -o NBody main.o CelestialBody.o $(CFLAGS)

main.o: main.cpp
	$(CC) -c main.cpp

CelestialBody.o: CelestialBody.cpp CelestialBody.h
	$(CC) -c CelestialBody.cpp

clean:
	rm $(OBJECTS) NBody
```

main.cpp

```
/**********************************************************
* @file main.cpp
* @author: Chris Olah
* @copyright Umass Lowell February 14 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 2A
* Professor: Dr. Daly
**********************************************************/

#include "CelestialBody.h"
#include "Universe.h"

using namespace std;

int main(int argc, char* argv[])
{
    Universe universe;

    int numofPlanets;
    cin >> numofPlanets;

    double radiusofUni;
    cin>>radiusofUni;

    for(int i = 0; i < numofPlanets; i++)
    {
        shared_ptr<CelestialBody>planet(new CelestialBody);
        cin >> *planet;
        planet->radius(radiusofUni);
        planet->position();
        universe.body.push_back(planet);
    }

    sf::RenderWindow window(sf::VideoMode(height,width),"Celestial Bodies");

    sf::Texture background;
    background.loadFromFile("starfield.jpg");

    sf::Sprite spritex;
    spritex.setTexture(background);
    spritex.setPosition(sf::Vector2f(0,0));


    while(window.isOpen())
    {
        sf:: Event event;
        while(window.pollEvent(event))
        {
            if(event.type == sf::Event::Closed)
            {
                window.close();
            }
    }
    window.clear();
    window.draw(spritex);

    for(auto const& a: universe.body)
    {
        window.draw(*a);
    }
    window.display();
    }
    return 0;
}

```

CelestialBody.h

```
/***********************************************************
* @file CelestialBody.h
* @author: Chris Olah
* @copyright Umass Lowell February 14 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 2A
* Professor: Dr. Daly
***********************************************************/

#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <memory>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

using namespace std;

const int height = 500;
const int width = 500;



class CelestialBody : public sf::Drawable
{
	public:
		CelestialBody();
		CelestialBody(double xposition, double yposition, double xvelocity, double yvelocity, double mass, double radius, string filename);
		friend istream& operator>> (istream &input, CelestialBody &b);
		void radius (float radius);
		void position();



	private:

    void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;

    double _xposition, _yposition;
    double _xvelocity, _yvelocity;
    double _mass;
    double _radius;
    string _filename;

    sf::Image image;
    sf::Sprite sprite;
    sf::Texture texture;
};
```

Universe.h

```
/**********************************************************
* @file Universe.h
* @author: Chris Olah
* @copyright Umass Lowell February 14 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 2A
* Professor: Dr. Daly
**********************************************************/

class Universe
{
	public:
		vector<shared_ptr<CelestialBody> >body;

	private:

};
```

CelestialBody.cpp

```
/**********************************************************
* @file CelestialBody.cpp
* @author: Chris Olah
* @copyright Umass Lowell February 14 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 2A
* Professor: Dr. Daly
**********************************************************/

#include "CelestialBody.h"
using namespace std;

CelestialBody::CelestialBody()
{
	return;
}

CelestialBody::CelestialBody(double xposition, double yposition, double xvelocity, double yvelocity, double mass, double radius, string filename)
{
	_xposition = xposition;
	_yposition = yposition;
	_xvelocity = xvelocity;
	_yvelocity = yvelocity;
	_mass = mass;
	_filename = filename;

	if (!image.loadFromFile(filename))
	{
		return;
	}

	texture.loadFromImage(image);
	sprite.setTexture(texture);
	sprite.setPosition(sf::Vector2f(_xposition, _yposition));
}

void CelestialBody::radius(float radius){
	_radius = radius;
	return;
}

void CelestialBody::position()
{
	double halfHeight = height/2;
	double halfWidth = width/2;
	double sfmlPos_x = ((_xposition/_radius) * halfWidth);
	double sfmlPos_y = ((_yposition/_radius) * halfHeight);
	_xposition = sfmlPos_x + halfWidth;
	_yposition = sfmlPos_y + halfHeight;

	sprite.setPosition(sf::Vector2f(_xposition, _yposition));
}

void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
	target.draw(sprite);
}

istream& operator>> (istream &input, CelestialBody &b)
{
	input >> b._xposition >> b._yposition >> b._yvelocity >> b._yvelocity >> b._mass >> b._filename;
	if (!b.image.loadFromFile(b._filename))
	{
		return input;
	}
	b.texture.loadFromImage(b.image);
	b.sprite.setTexture(b.texture);
	b.sprite.setPosition(sf::Vector2f(b._xposition, b._yposition)); //intial

	return input;
}

```

PS2b Adding Physics Simulation and Animation

The Assignment

For part B, we had to succeed in loading and displaying a static universe. We were once again provided with images and input files that we used to create the universe like the last assignment. Additionally, we had to add simulation, so the planets behave like real planets. They had to move at the correct velocity with respect to other planets to create a realistic simulation. We had to use Newton's Laws to of motion and gravitation to find the net force exerted on each particle in order to model the dynamics of the system. We had to utilize the leapfrog finite difference approximation scheme to obtain the required values.

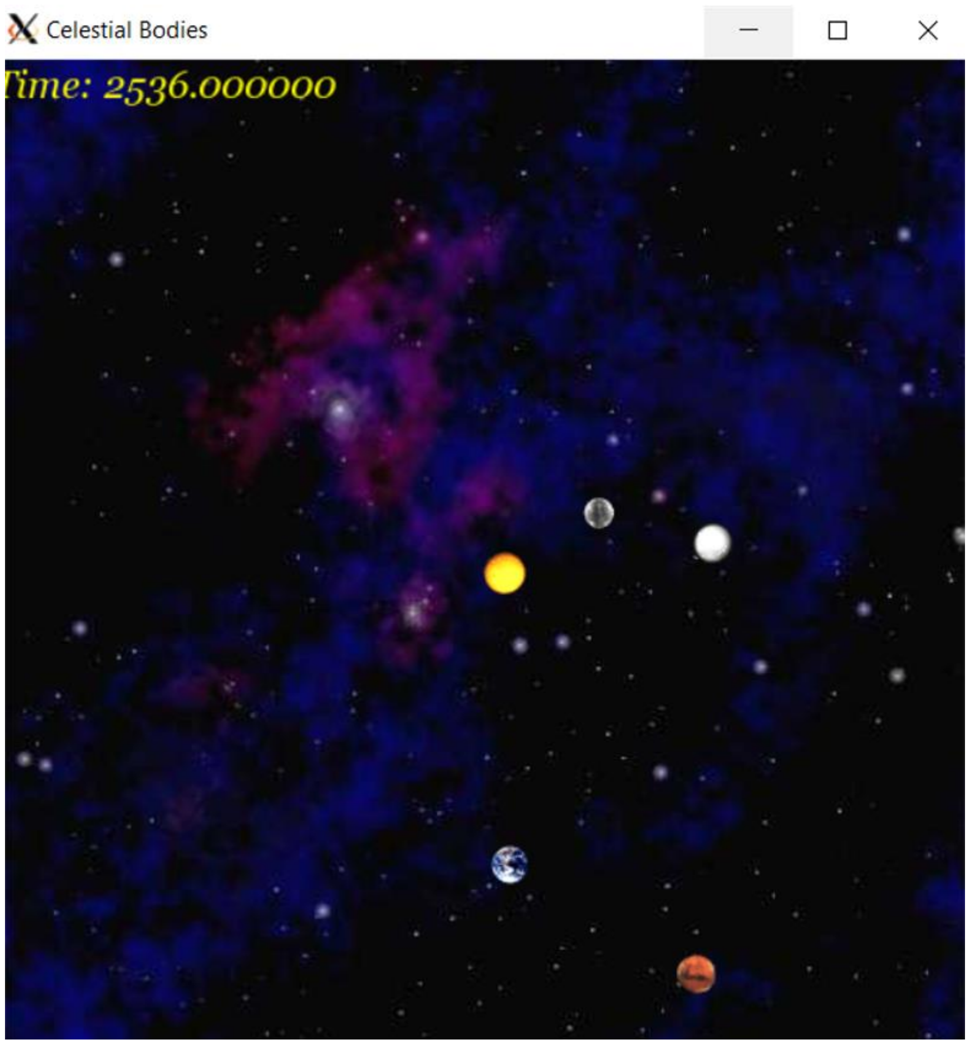
Key Concepts

Finding the positions of the planets was important. I took in the pos from the input and then divided it by the radius to make the large numbers smaller and easier to work with. I then multiplied it by either the height or the width, depending on the x or y position, in order to get the position in SFML coordinate system. Because in SFML, 0,0 is top left and not the middle. I also overloaded the >> operator which used istream to take input from cin and pass it into the variables. Using the lecture and the information of the assignment page, I used the formulas given to get the right equations in order to compute the forces, velocity and acceleration. I divided the forces into x and y and later put it together to get the net forces. I used pow() and sqrt() to get the R. The physics formulas given to us were a key factor as they determined the right values being used to simulate the universe.

What I Learned

I learned a lot more about physics in this assignment and how to create more complicated functions in our programs in order to be accurate. I also got a glimpse of how audio can be used in the SFML environment as we had background music for this window.

Output/Screenshot



PS2b Source Code:

Makefile

```
CC= g++ -std=c++17
CFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
OBJECTS = main.o CelestialBody.o

all:
	make NBody

NBody:$(OBJECTS)
	$(CC) -o NBody main.o CelestialBody.o $(CFLAGS)

main.o: main.cpp
	$(CC) -c main.cpp

CelestialBody.o: CelestialBody.cpp CelestialBody.h
	$(CC) -c CelestialBody.cpp

clean:
	rm $(OBJECTS) NBody
```

main.cpp

```
/************************************************************
* @file main.cpp
* @author: Chris Olah
* @copyright Umass Lowell February 22 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 2B
* Professor: Dr. Daly
************************************************************/

#include "CelestialBody.h"
using namespace std;

int main(int argc, char* argv[])
{
    sf::RenderWindow window(sf::VideoMode(512,512),"Celestial Bodies");
    Universe universe;

    double universe_time = atof (argv[1]); //get values from command line
    double step_time = atof ( argv[2]);
    double simulation = 0;

    int numofPlanets;
    cin >> numofPlanets;

    double radiusofUni;
    cin >> radiusofUni;

    vector <CelestialBody> trial;

    for(int i = 0; i < numofPlanets; i++) //get planets
   {
        shared_ptr<CelestialBody>planet(new CelestialBody);
        cin >> *planet;
        planet->radius(radiusofUni);
        planet->position();
        trial.push_back(*planet);
        universe.body.push_back(planet);
    }

    sf::Music music;//play background music
    if(!music.openFromFile("2001.wav"))
    {
        return -1;
    }
    music.play();


    sf::Font time_font; //set font
    time_font.loadFromFile("Georgia Italic.ttf");

    sf::Text text;
    text.setFont(time_font);
    text.setCharacterSize(20);
    text.setFillColor(sf::Color::Yellow);

    sf::Texture background; // set the background
    background.loadFromFile("starfield.jpg");

   sf::Sprite spritex;
   spritex.setTexture(background);

   vector<CelestialBody>::iterator it;
    vector<CelestialBody>::iterator x, y;


    while(window.isOpen())
    {
             sf:: Event event;
             while(window.pollEvent(event))
             {
                     if(event.type == sf::Event::Closed)
                     {
                             window.close();
                     }
             }

    window.clear();
    window.draw(spritex);

    text.setString("Time: " + to_string(simulation));
    window.draw(text);
```

```

    double fx, fy;
    x = trial.begin();

    for(int i = 0; i < numofPlanets; i++)
    {
        y = trial.begin();
        fx = 0;
        fy = 0;

        for(int j = 0; j < numofPlanets; j++)
        {
            if(i != j)
            {
                fx += universe.calc_force_x(*x,*y);
                fy += universe.calc_force_y(*x,*y);
            }
                    y++;
                }

            x->forces(fx,fy);
            x++;
        }

        for(it = trial.begin(); it != trial.end(); it++)
        {
            window.draw(*it);
            it->step(step_time);
            it->position();
        }

        window.display();
        simulation++;
        if(simulation >= universe_time)
        {
            cout << "hello";
        }
    }
    return 0;
}

```

CelestialBody.h

```
/**********************************************************
* @file CelestialBody.h
* @author: Chris Olah
* @copyright Umass Lowell February 22 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 2B
* Professor: Dr. Daly
**********************************************************/

#ifndef CelestialBody_H
#define CelestialBody_H

#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <cmath>
#include <vector>
#include <memory>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>

using namespace std;
const int height = 512;
const int width = 512;

class CelestialBody:public sf::Drawable
{
	public:

		CelestialBody();//default
		CelestialBody(double xposition, double yposition, double xvelocity, double yvelocity, double mass, string filename); // Constructor
		friend istream& operator>> (istream &input, CelestialBody &b);//overloaded
		void radius (float radius);
		void position();

```

```
        void forces (double xForce, double yForce);//used
        void step (double time);

        double get_mass()
        {
             return _mass;
        }

        double get_x()
        {
            return _xposition;
        }

        double get_y()
        {
            return _yposition;
        }

    private:

        void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;

        double _xposition, _yposition;
        double _xvelocity, _yvelocity;
        double _mass;
        double _radius;
        string _filename;


        double acceleration_x;
        double acceleration_y;

        double force_x;
        double force_y;

        sf::Image image;
        sf::Sprite sprite;
        sf::Texture texture;

};

class Universe
```

```
{
	public:
		vector<shared_ptr<CelestialBody> >body;

		double calc_force_x (CelestialBody &first, CelestialBody &second);
		double calc_force_y (CelestialBody &first, CelestialBody &second);

	private:
		double seconds;
		double net_force;

};

#endif
```

CelestialBody.cpp

```
#include "CelestialBody.h"

CelestialBody::CelestialBody()
{
	return;
}

CelestialBody::CelestialBody(double xposition, double yposition, double xvelocity, double yvelocity,
double mass,string filename)
{
	_xposition = xposition;
	_yposition = yposition;
	_xvelocity = xvelocity;
	_yvelocity = yvelocity;
	_mass = mass;
	_filename = filename;
	texture.loadFromImage(image);
	sprite.setTexture(texture);
	sprite.setPosition(sf::Vector2f(_xposition, _yposition));
}

istream& operator>> (istream &input, CelestialBody &b)
{
	input >> b._xposition >> b._yposition;
	input >> b._yvelocity >> b._yvelocity;
	input >> b._mass >> b._filename;

	b.texture.loadFromFile(b._filename);
	b.sprite.setTexture(b.texture);
	b.sprite.setPosition(sf::Vector2f(b._xposition, b._yposition)); //intial

	b.acceleration_x = 0;
	b.acceleration_y = 0;

	b.force_x = 0;
	b.force_y = 0;

	return input;
}

```

```
void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
	target.draw(sprite);
}


void CelestialBody::radius(float radius)
{
	_radius = radius;
}

void CelestialBody::forces(double xForce, double yForce)
{
	force_x = xForce;
	force_y = yForce;
}


void CelestialBody::position()//done
{
	double newxposition = ((_xposition / _radius) * (width / 2)) + (height / 2);
	double newyposition = ((_yposition / _radius) * (width / 2)) + (height / 2);

	sprite.setPosition(sf::Vector2f(newyposition, newxposition));
}

double Universe::calc_force_x(CelestialBody& first, CelestialBody& second)
{
	const double G = 6.67e-11;
	double delta_x = second.get_x() - first.get_x();
	double delta_y = second.get_y() - first.get_y();
	double r = pow(delta_x,2) + pow (delta_y, 2);
	double sqrtr = sqrt(r);

	double f = ( G * first.get_mass() * second.get_mass()) / r;
	double F = f* (delta_x /sqrtr);

	return F;

}

double Universe::calc_force_y(CelestialBody& first, CelestialBody& second)
```

```
{
    const double G = 6.67e-11;

    double delta_x = second.get_x() - first.get_x();
    double delta_y = second.get_y() - first.get_y();
    double r = pow(delta_x,2) + pow (delta_y, 2);
    double sqrtr = sqrt(r);

    double f = ( G * first.get_mass() * second.get_mass()) / r;
    double F = f* (delta_y /sqrtr);

    return F;

}

void CelestialBody::step(double t)
{

    acceleration_x = force_x / _mass;
    acceleration_y = force_y / _mass;

    _xvelocity = _xvelocity + (acceleration_x * t);
    _yvelocity = _yvelocity + (acceleration_y * t);

    _xposition = _xposition + ( _xvelocity * t);
    _yposition = _yposition + ( _yvelocity * t);

}

```

PS3 Recursive Graphics

The Assignment

The purpose of the assignment was for the program to take in instructions and print out a Pythagorean tree. The instructions were the size of the base square and the number of times you want the recursion to occur. Given the length (L) we created an SFML window with dimensions of 6L x 4L. We implemented the program to begin with a rectangular block and continue to create smaller structures stemming from the original block using simple mathematical functions.

Key Concepts

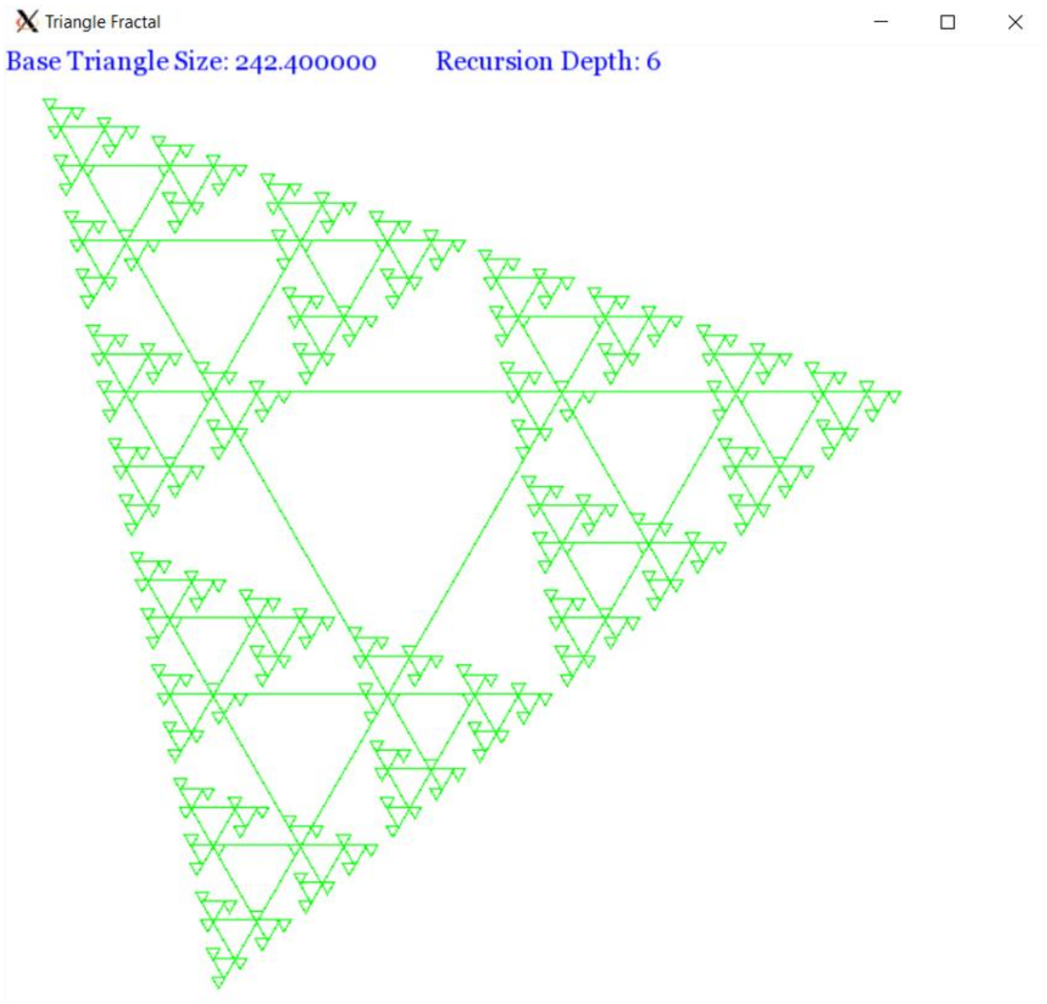
One of the key components of the assignment was using ConvexShape in order to draw the square. We gave it four coordinates to draw the square and we stored it in a vector. Additionally, pTree called itself to draw the sqaure recursively and it kept calling itself till "depths" or the number of recursions ran out and calculated the squares positioning using trigonometric equations. The following code segment displays the new triangle calculations:

```
case 1:
        tri.move(-length/2, -(length/2 * sqrt(3)));
        break;
case 2:
        tri.move(length*2, 0);
        break;
case 3:
        tri.move(0, length * sqrt(3));
        break;
```

What I Learned

I learned to critically think about recursive algorithm. Recursion has always been little tricky for me and this assignment really helped me get more comfortable with it. Additionally, I got more practice with animation within SFML as I created a shape that changes color as a response to keystroke. I also learned to add color and alter the shape of the Pythagorean tree by altering its angles.

Output/Screenshot



Base Triangle Size: 242.400000   Recursion Depth: 6

PS3 Source Code:

Makefile

```
CC = g++

all: TFractal

TFractal: Triangle.o TFractal.o
	g++ Triangle.o TFractal.o -o TFractal -lsfml-graphics -lsfml-window -lsfml-system

Triangle.o: Triangle.cpp Triangle.h
	g++ -c Triangle.cpp -Wall -Werror -pedantic

TFractal.o: TFractal.cpp
	g++ -c TFractal.cpp -Wall -Werror -pedantic
clean:
	rm *.o TFractal
```

TFractal.cpp

```
/**********************************************************
* @file TFractal.cpp
* @author: Chris Olah
* @copyright Umass Lowell February 28 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 3 Triangle Fractal
* Professor: Dr. Daly
**********************************************************/

#include "TFractal.h"
#include "Triangle.h"

int main(int argc, char* argv[]) {
	double L = atof(argv[1]);
	int N = atoi(argv[2]); //< Take command line arguments
	int windowSize = L * 3; //< Set window size
	Triangle myTriangle(L, windowSize);

	sf::Font font;
	if (!font.loadFromFile("Georgia.ttf"))
		return -1;
	sf::Text text;
	text.setFont(font);
	text.setString("Base Triangle Size: " + std::to_string(L) \
	+ " Recursion Depth: " + std::to_string(N));
	text.setCharacterSize(18);

	sf::RenderWindow window1(sf::VideoMode(L * 3, L *3), "Triangle Fractal");
	while (window1.isOpen()) {
		sf::Event event;
		while (window1.pollEvent(event)) {
			if (event.type == sf::Event::Closed)
				window1.close();
		}
		window1.clear(sf::Color::White);
		fTree(window1, L, N, windowSize); //< call fTree window function
		window1.draw(text);
		window1.display();
	}
	return 0;
}


void fTree(sf::RenderTarget& target, int N, Triangle& parent) { // NOLINT
	if (N < 1) {
		return;
	}

	target.draw(parent); //< Draw base triangle

	double newLength = (parent.getLength() / 2); //< Store new triangle length

	Triangle child1(newLength);
	Triangle child2(newLength);
	Triangle child3(newLength); //< Create new child triangles

	child1.setPosition(parent.getPosition());
	child2.setPosition(parent.getPosition());
	child3.setPosition(parent.getPosition());

	child1.reposition(1, newLength);
	child2.reposition(2, newLength);
	child3.reposition(3, newLength);

	fTree(target, N - 1, child1);
	fTree(target, N - 1, child2);
	fTree(target, N - 1, child3);
}

void fTree(sf::RenderTarget& target, double length, int N, int windowSize) { //NOLINT
	Triangle myTriangle(length, windowSize);
	fTree(target, N, myTriangle);
}

```

TFractal.h

```
/*********************************************************
* @file TFractal.h
* @author: Chris Olah
* @copyright Umass Lowell February 28 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 3
* Professor: Dr. Daly
*********************************************************/
#ifndef _HOME_CHRIS_OLAH_PS3_TFRACTAL_H_
#define _HOME_CHRIS_OLAH_PS3_TFRACTAL_H_

#include "Triangle.h"

#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

void fTree(sf::RenderTarget& target, int recursionDepth, const Triangle& parent); // NOLINT
void fTree(sf::RenderTarget& target, double triLength, int recursionDepth, int windowSize); // NOLINT

#endif // _HOME_CHRIS_OLAH_PS3_TFRACTAL_H_
```

Triangle.h

```
/**********************************************************
 * @file Triangle.h
 * @author: Chris Olah
 * @copyright Umass Lowell February 28 2022
 * @version 1.0
 * Course: Computing IV (COMP.2040)
 * Assignment: Project 3 Triangle Fractal
 * Professor: Dr. Daly
 **********************************************************/

#ifndef _HOME_CHRIS_OLAH_PS3_TRIANGLE_H_
#define _HOME_CHRIS_OLAH_PS3_TRIANGLE_H_

#include <iostream>
#include <cmath>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

class Triangle : public sf::Drawable {
	public:
		explicit Triangle(double sideLength);
		Triangle(double sideLength, int windowSize);
		Triangle(const Triangle &t);
		double getLength() {
			return length;
		}
		sf::Vector2f getPosition(void);
		void setColor(const Triangle &t);
		void setPosition(sf::Vector2f initialPos);
		void reposition(int childNum, double length);
	private:
		double length;
		sf::ConvexShape tri;
		virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const {
			target.draw(tri, states);
		}
};
#endif // _HOME_CHRIS_OLAH_PS3_TRIANGLE_H_
```

Triangle.cpp

```
/**********************************************************
* @file Triangle.cpp
* @author: Chris Olah
* @copyright Umass Lowell February 28 2022
* @version 1.0
* Course: Computing IV (COMP.2040)
* Assignment: Project 3 Triangle Fractal
* Professor: Dr. Daly
**********************************************************/

#include "Triangle.h"
#include <random>

Triangle::Triangle(double sideLength, int windowSize) {
	tri.setPointCount(3);
	tri.setPoint(0, sf::Vector2f(0, 0));
	tri.setPoint(1, sf::Vector2f(sideLength/2, (sideLength/2) * sqrt(3)));
	tri.setPoint(2, sf::Vector2f(sideLength, 0));

	tri.setOutlineThickness(1);
	if (sideLength > 60)
		tri.setOutlineColor(sf::Color::Green);
	else
		tri.setOutlineColor(sf::Color::Blue);
		tri.setFillColor(sf::Color::Transparent);
		tri.setPosition(windowSize/5, windowSize/3);

		length = sideLength;
}

Triangle::Triangle(const Triangle &t) {
	tri = t.tri;
}

Triangle::Triangle(double sideLength) {
	tri.setPointCount(3);
	tri.setPoint(0, sf::Vector2f(0, 0));
	tri.setPoint(1, sf::Vector2f(sideLength/2, (sideLength/2) * sqrt(3)));
	tri.setPoint(2, sf::Vector2f(sideLength, 0));

```

```
    tri.setOutlineThickness(1);
    tri.setOutlineColor(sf::Color::Blue);
    tri.setFillColor(sf::Color::Transparent);

    length = sideLength;
}

sf::Vector2f Triangle::getPosition(void) {
    return tri.getPosition();
}

void Triangle::setPosition(sf::Vector2f initialPos) {
    tri.setPosition(initialPos);
    return;
}

void Triangle::setColor(const Triangle &t) {
    tri.setOutlineColor(sf::Color(rand() % 255, rand() % 255, rand() % 255));
}

void Triangle::reposition(int childNum, double length) {
    switch (childNum) {
        case 1:
            tri.move(-length/2, -(length/2 * sqrt(3)));
            break;
        case 2:
            tri.move(length*2, 0);
            break;
        case 3:
            tri.move(0, length * sqrt(3));
            break;
    }
}
```

PS4a CircularBuffer Implementation with Unit Tests + Exceptions

The Assignment

The assignment was to write a program to simulate plucking a guitar string using the Karplus-Strong algorithm and learn about cpplint while using Unit testing and exceptions. The first part was focused on imitating the Karplus-Strong algorithm and utilizing cpplint to check our code for the appropriate format. The Karplus-Strong algorithm simulates the vibration of an instrument by maintaining a ring buffer of N samples. The algorithm repeatedly deletes the first sample from the buffer and adds to the end of the buffer the average of the deleted sample and the first samples. Our goal was to implement this algorithm.

Key Concepts

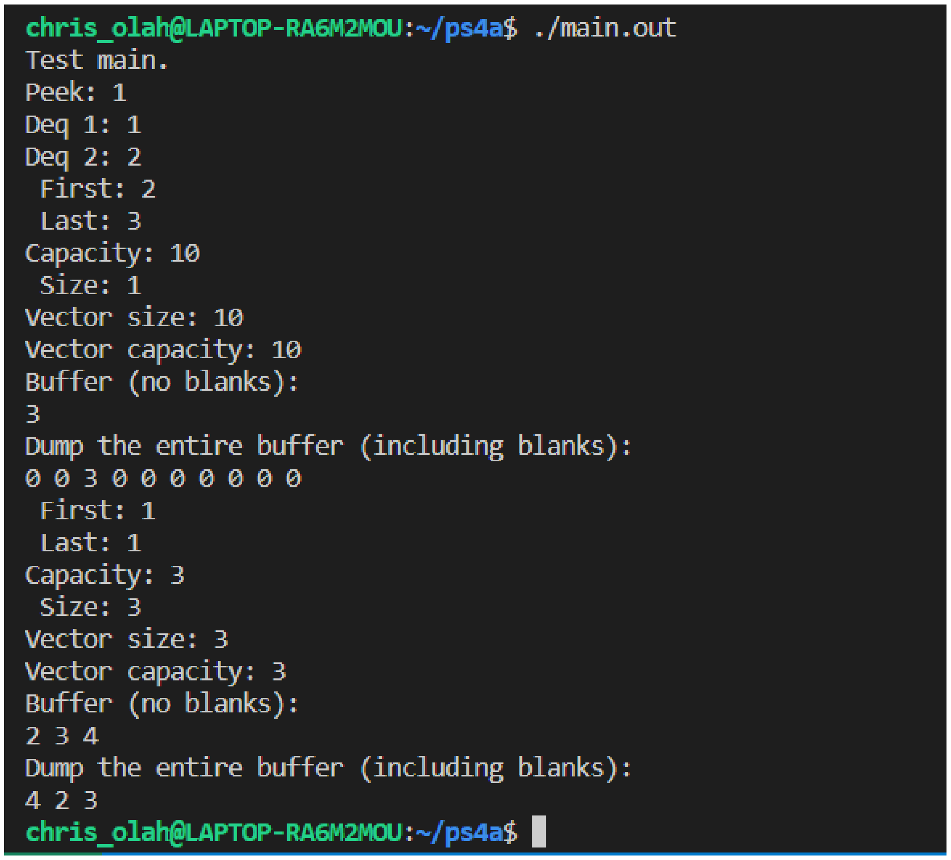
The main concepts for this assignment focused on implementing the RingBuffer using exceptions, which is something we had not really discussed in class before this assignment. Try / Catch blocks were used to test for invalid actions. Here’s an instance of an exception thrown in the constructor:

```
CircularBuffer::CircularBuffer(size_t capacity) {
    if (capacity < 1) {
        throw std::invalid_argument("capacity must be greater than zero");
    }
```

What I Learned

I learned about exceptions and cpplint in this assignment. Cpplint was a new format and threw me off at first but I adjusted to its format. It was interesting to see what errors it caught which was easily ignored by me. I also learned about the Karplus-Strong algorithm, which was tricky but also fun to learn.

Output/Screenshot



```
chris_olah@LAPTOP-RA6M2MOU:~/ps4a$ ./main.out
Test main.
Peek: 1
Deq 1: 1
Deq 2: 2
 First: 2
 Last: 3
Capacity: 10
 Size: 1
Vector size: 10
Vector capacity: 10
Buffer (no blanks):
3
Dump the entire buffer (including blanks):
0 0 3 0 0 0 0 0 0 0
 First: 1
 Last: 1
Capacity: 3
 Size: 3
Vector size: 3
Vector capacity: 3
Buffer (no blanks):
2 3 4
Dump the entire buffer (including blanks):
4 2 3
chris_olah@LAPTOP-RA6M2MOU:~/ps4a$
```

PS4a Source Code:

Makefile

```
CC= g++ -std=c++17
CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
Boost= -lboost_unit_test_framework

all: main.out ps4a

ps4a: test.o CircularBuffer.o
	$(CC) test.o CircularBuffer.o -o ps4a $(Boost)

test.o: test.cpp CircularBuffer.hpp
	$(CC) -c test.cpp CircularBuffer.hpp $(CFLAGS)

main.out: main.o CircularBuffer.o
	$(CC) main.o CircularBuffer.o -o main.out

main.o: main.cpp CircularBuffer.hpp
	$(CC) -c main.cpp $(CFLAGS)

CircularBuffer.o: CircularBuffer.cpp CircularBuffer.hpp
	$(CC) -c CircularBuffer.cpp $(CFLAGS)

clean:
	rm *.o
	rm *.gch
	rm ps4a
	rm *.out
```

main.cpp

```cpp
// Copyright 2022 Chris Olah
#include "CircularBuffer.hpp"

int main() {
    std::cout << "Test main.\n";

     CircularBuffer test(10);
    test.enqueue(1);
    test.enqueue(2);
    test.enqueue(3);
    std::cout << "Peek: " << test.peek() << "\n";

    std::cout << "Deq 1: " << test.dequeue() << "\n";
    std::cout << "Deq 2: " << test.dequeue() << "\n";

    test.output();

    // Test looping back around
    CircularBuffer test2(3);

    test2.enqueue(1);
    test2.enqueue(2);
    test2.enqueue(3);

    test2.dequeue();
    test2.dequeue();
    test2.dequeue();

    test2.enqueue(1);
    test2.enqueue(2);
    test2.enqueue(3);

    test2.dequeue();
    test2.enqueue(4);

    test2.output();

    return 0;
}
```

CircularBuffer.hpp

```
// Copyright 2022 Chris Olah

#ifndef _HOME_CHRIS_OLAH_PS4A_CIRCULARBUFFER_HPP_
#define _HOME_CHRIS_OLAH_PS4A_CIRCULARBUFFER_HPP_

#include <stdint.h>
#include <iostream>
#include <string>
#include <vector>
#include <stdexcept>
#include <exception>
#include <sstream>

class CircularBuffer {
	public:
		// create an empty ring buffer, with given max capacity
		explicit CircularBuffer(size_t capacity);
		size_t size(); // return number of items currently in the buffer
		bool isEmpty(); // is the buffer empty (size equals zero)?
		bool isFull(); // 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 output(); // output process

	private:
		std::vector<int16_t> buf;
		int first;
		int last;
		int cap;
		int s;
};

#endif // _HOME_CHRIS_OLAH_PS4A_CIRCULARBUFFER_HPP_
```

CircularBuffer.cpp

```
// Copyright 2022 Chris Olah
#include "CircularBuffer.hpp"

CircularBuffer::CircularBuffer(size_t capacity) {
	if (capacity < 1) {
		throw std::invalid_argument("capacity must be greater than zero");
	}

	last = 0;
	first = 0;
	s = 0;
	cap = capacity;
	buf.resize(capacity);

	return;
}

bool CircularBuffer::isEmpty() {
	if (s != 0) {
		return false;
	} else {
		return true;
	}
}

bool CircularBuffer::isFull() {
	if (s == cap) {
		return true;
	} else {
		return false;
	}
}

size_t CircularBuffer::size() {
	return s;
}

void CircularBuffer::enqueue(int16_t x) {
	if (isFull()) {
		throw std::runtime_error("enqueue: can’t enqueue to a full ring");
```

```
    }
    if (last >= cap) {
        last = 0;
    }
    buf.at(last) = x; // keep going
    last++;
    s++;
}

int16_t CircularBuffer::dequeue() {
    if (isEmpty()) {
        throw std::runtime_error("dequeue: can’t dequeue an empty ring");
    }
    int16_t first16 = buf.at(first);
    buf.at(first) = 0;
    first++;
    s--;
    if (first >= cap) {
        first = 0;
    }
    return first16;
}

int16_t CircularBuffer::peek() {
    if (isEmpty()) {
        throw std::runtime_error("peek: can’t peek an empty ring");
    }
    return buf.at(first);
}

void CircularBuffer::output() {
    std::cout << " First: " << first << "\n";
    std::cout << " Last: " << last << "\n";
    std::cout << "Capacity: " << cap << "\n";
    std::cout << " Size: " << s << "\n";
    std::cout << "Vector size: " << buf.size() << "\n";
    std::cout << "Vector capacity: " << buf.capacity() << "\n";
    std::cout << "Buffer (no blanks): \n";

    int x = 0;
    int y = first;

    while (x < s) { // Make the loop go back to 0 to continue printing.
        if (y >= cap) {
            y = 0;
        }

        std::cout << buf[y] << " ";
        y++;
        x++;
    }

    std::cout << "\nDump the entire buffer (including blanks): \n";

    for (int x = 0; x < cap; x++) {
        std::cout << buf[x] << " ";
    }

    std::cout << "\n";
}
```

test.cpp

```
// Copyright 2022 Chris Olah

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

// Tests various aspects of the contructor.
BOOST_AUTO_TEST_CASE(Constructor) {
	// Normal constructor - shouldn’t fail.
	BOOST_REQUIRE_NO_THROW(CircularBuffer(100));

	BOOST_REQUIRE_THROW(CircularBuffer(0), std::invalid_argument);
}

// Checks the size() method
BOOST_AUTO_TEST_CASE(Size) {
	CircularBuffer test(1);
	// This should be size 0.
	BOOST_REQUIRE(test.size() == 0);
	test.enqueue(5);
	// This should be size 1.
	BOOST_REQUIRE(test.size() == 1);
	test.dequeue();
	BOOST_REQUIRE(test.size() == 0);
}

// Checks the isEmpty() method
BOOST_AUTO_TEST_CASE(isEmpty) {
	// This should be true
	CircularBuffer test(5);
	BOOST_REQUIRE(test.isEmpty() == true);
	// This should be false

	CircularBuffer test2(5);
	test2.enqueue(5);
	BOOST_REQUIRE(test2.isEmpty() == false);
}

// Checks the isFull() method
```

```
BOOST_AUTO_TEST_CASE(isFull) {
	CircularBuffer test(5);
	BOOST_REQUIRE(test.isFull() == false);
	CircularBuffer test2(1);
	test2.enqueue(5);
	BOOST_REQUIRE(test2.isFull() == true);
}

// Test enqueue
BOOST_AUTO_TEST_CASE(Enqueue) {
	// These test basic enqueuing
	CircularBuffer test(5);
	BOOST_REQUIRE_NO_THROW(test.enqueue(1));
	BOOST_REQUIRE_NO_THROW(test.enqueue(2));
	BOOST_REQUIRE_NO_THROW(test.enqueue(3));
	BOOST_REQUIRE_NO_THROW(test.enqueue(4));
	BOOST_REQUIRE_NO_THROW(test.enqueue(5));
	BOOST_REQUIRE_THROW(test.enqueue(6), std::runtime_error);
}

// Test dequeue
BOOST_AUTO_TEST_CASE(Dequeue) {
	CircularBuffer test(5);
	test.enqueue(0);
	test.enqueue(1);
	test.enqueue(2);
	BOOST_CHECK_NO_THROW(test.peek());
	BOOST_REQUIRE(test.dequeue() == 0);
	BOOST_REQUIRE(test.dequeue() == 1);
	BOOST_REQUIRE(test.dequeue() == 2);

	BOOST_REQUIRE_THROW(test.dequeue(), std::runtime_error);
}
```

PS4b StringSound Implementation and SFML Audio Output

The Assignment

In the second part (PS4b), we used the RingBuffer from PS4a to create a model of a Guitar, by implementing the Karplus-Strong algorithm to simulate the plucking of a guitar string. We had to generate a stream of string samples for audio playback under keyboard control. Using the audio resources available through SFML we had to sync the keys on the keyboard (“q2we4r5ty 7u8i9op-[=zxdcfvgbnjmk,.;/'”) to those of the musical notes given to us in a certain order (37 notes from 110Hz to 880 Hz). Once again, we had to use C++ exceptions for error handling.

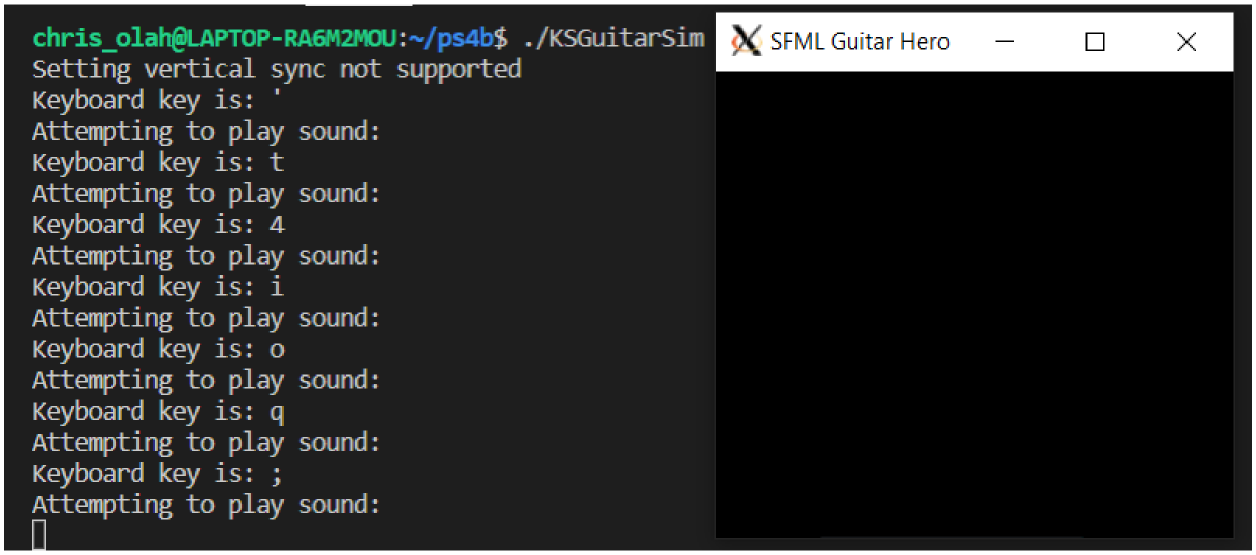
Key Concepts

In this part of the assignment generating audio was the most important part. There are two parts that are involved in doing so which is getting values out of StringSound object and into SFML audio playback object and playing the audio object when key press events occur. We used SFML to create an electronic keyboard and handled the events using sf::Event::KeyPressed. A StringSound object is created for each frequency and each of those produces audio samples that are loaded into a sf::Int16 vector. These vectors are made into sf:: SoundBuffers which are then made into sf::Sound objects. Later in an event loop within the SFML window the sounds are played when a key is pressed.

What I Learned

I got more experience with vectors as we dealt with sf::Int16 vectors to produce the sounds. Also, we played around with our code to obtain different instruments which helped us gain a better sense of the program and what it was doing.

Output/Screenshot



```
chris_olah@LAPTOP-RA6M2MOU:~/ps4b$ ./KSGuitarSim
Setting vertical sync not supported
Keyboard key is: '
Attempting to play sound:
Keyboard key is: t
Attempting to play sound:
Keyboard key is: 4
Attempting to play sound:
Keyboard key is: i
Attempting to play sound:
Keyboard key is: o
Attempting to play sound:
Keyboard key is: q
Attempting to play sound:
Keyboard key is: ;
Attempting to play sound:
```

PS4b Source Code:

Makefile

```
CC= g++ -std=c++17
CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
SFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
Boost= -lboost_unit_test_framework

        all: KSGuitarSim

KSGuitarSim: KSGuitarSim.o StringSound.o CircularBuffer.o
	$(CC) KSGuitarSim.o StringSound.o CircularBuffer.o -o KSGuitarSim $(SFLAGS)

KSGuitarSim.o: KSGuitarSim.cpp StringSound.hpp
	$(CC) -c KSGuitarSim.cpp StringSound.hpp $(CFLAGS)

StringSound.o: StringSound.cpp StringSound.hpp
	$(CC) -c StringSound.cpp StringSound.hpp $(CFLAGS)

CircularBuffer.o: CircularBuffer.cpp CircularBuffer.hpp
	$(CC) -c CircularBuffer.cpp CircularBuffer.hpp $(CFLAGS)

clean:
	rm *.o
	rm *.gch
	rm KSGuitarSim
```

KSGuitarSim.cpp

```
// Copyright 2022 Chris Olah

#include <math.h>
#include <limits.h>
#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include <vector>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>
#include "StringSound.hpp"

#define CONCERT_A 220.0 // 220 for piano and guitar //520 for drum
#define SAMPLES_PER_SEC 44100 // 4100 for drum
const int keyboard_size = 37;

std::vector<sf::Int16> makeSamplesFromString(StringSound gs) {
	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;
}

int main() {
	sf::RenderWindow window(sf::VideoMode(800, 800), "SFML Guitar Hero");
	sf::Event event;

	double frequency;
	std::vector<sf::Int16> sample;
```

```

    std::vector<std::vector<sf::Int16>> samples(keyboard_size);
    std::vector<sf::SoundBuffer> buffers(keyboard_size);
    std::vector<sf::Sound> sounds(keyboard_size);

    std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/’ ";

    for (int i = 0; i < (signed)keyboard.size(); i++) {
        // 24 for guitar //8 for piano //24 for guitar
        frequency = CONCERT_A * pow(2, ((i - 24) / 12.0));
        StringSound tmp = StringSound(frequency);

        sample = makeSamplesFromString(tmp);
        samples[i] = sample;

        if (!buffers[i].loadFromSamples(&samples[i][0], \
        samples[i].size(), 2, SAMPLES_PER_SEC)) {
            throw std::runtime_error\
            ("sf::SoundBuffer: failed to load from samples.");
        }

        sounds[i].setBuffer(buffers[i]);
    }

    while (window.isOpen()) {
        while (window.pollEvent(event)) {
        if (event.type == sf::Event::TextEntered) {
            if (event.text.unicode < 128) {
                char key = static_cast<char>(event.text.unicode);

                for (int i = 0; i < (signed)keyboard.size(); i++) {
                    if (keyboard[i] == key) {
                        std::cout << "Keyboard key is: " \
                        << keyboard[i] << "\n";
                        std::cout << "Attempting to play sound:\n";
                        sounds[i].play();
                        break;
                    }
                }
            }
        }

```

```
            if (event.type == sf::Event::Closed) {
                window.close();
            }
        }

        window.clear();
        window.display();
    }
    return 0;
}
```

CircularBuffer.hpp

```
// Copyright 2022 Chris Olah

#ifndef _HOME_CHRIS_OLAH_PS4B_CIRCULARBUFFER_HPP_
#define _HOME_CHRIS_OLAH_PS4B_CIRCULARBUFFER_HPP_

#include <stdint.h>
#include <iostream>
#include <string>
#include <vector>
#include <stdexcept>
#include <exception>
#include <sstream>

class CircularBuffer {
    public:
        // create an empty ring buffer, with given max capacity
        explicit CircularBuffer(size_t capacity);
        size_t size(); // return number of items currently in the buffer
        bool isEmpty(); // is the buffer empty (size equals zero)?
        bool isFull(); // 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
        int getCapacity() {
            return cap;
        }
        int getSize() {
            return s;
        }

    private:
        std::vector<int16_t> buf;
        int first;
        int last;
        int cap;
        int s;
};

#endif // _HOME_CHRIS_OLAH_PS4B_CIRCULARBUFFER_HPP_
```

CircularBuffer.cpp

```
// Copyright 2022 Chris Olah
#include "CircularBuffer.hpp"

CircularBuffer::CircularBuffer(size_t capacity) {
	if (capacity < 1) {
		throw std::invalid_argument("capacity must be greater than zero");
	}

	last = 0;
	first = 0;
	s = 0;
	cap = capacity;
	buf.resize(capacity);

	return;
}

bool CircularBuffer::isEmpty() {
	if (s != 0) {
		return false;
	} else {
		return true;
	}
}

bool CircularBuffer::isFull() {
	if (s == cap) {
		return true;
	} else {
		return false;
	}
}

size_t CircularBuffer::size() {
	return s;
}

void CircularBuffer::enqueue(int16_t x) {
	if (isFull()) {
		throw std::runtime_error("enqueue: can’t enqueue to a full ring");
```

```
    }
    if (last >= cap) {
        last = 0;
    }
    buf.at(last) = x; // keep going
    last++;
    s++;
}

int16_t CircularBuffer::dequeue() {
    if (isEmpty()) {
        throw std::runtime_error("dequeue: can’t dequeue an empty ring");
    }
    int16_t first16 = buf.at(first);
    buf.at(first) = 0;
    first++;
    s--;
    if (first >= cap) {
        first = 0;
    }
    return first16;
}

int16_t CircularBuffer::peek() {
    if (isEmpty()) {
        throw std::runtime_error("peek: can’t peek an empty ring");
    }
    return buf.at(first);
}
```

StringSound.hpp

```
// Copyright 2022 Chris Olah

#ifndef _HOME_CHRIS_OLAH_PS4B_STRINGSOUND_HPP_
#define _HOME_CHRIS_OLAH_PS4B_STRINGSOUND_HPP_

#include <cmath>
#include <iostream>
#include <string>
#include <vector>
#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>

#include "CircularBuffer.hpp"

class StringSound {
    public:
        explicit StringSound(double frequency);
        // create a guitar string sound of the
        // given frequency using a sampling rate
        // of 44,100

        explicit 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
        sf::Int16 sample(); // return the current sample
        int time(); // return number of times tic was called
        // so far
    private:
        CircularBuffer* buffer;
        int tic_time;
        int num;
};
```

```


#endif // _HOME_CHRIS_OLAH_PS4B_STRINGSOUND_HPP_
```

StringSound.cpp

```
// Copyright 2022 Chris Olah

#include "StringSound.hpp"
#include <vector>

#define SAMPLING_RATE 44100

 StringSound::StringSound(std::vector<sf::Int16>init) {
        buffer = new CircularBuffer(init.size());
        if (buffer == NULL) {
              throw std::runtime_error\
              ("StringSound Constructor: Could not allocate memory");
        }

        std::vector<sf::Int16>::iterator it;

        for (it = init.begin(); it < init.end(); it++) {
              buffer->enqueue((int16_t)*it);
        }
        tic_time = 0;
}

void StringSound::pluck() {
        for (int i = 0; i < num; i++) {
              buffer->dequeue();
        }
        for (int i = 0; i < num; i++) {
              buffer->enqueue((rand() % 32768)); // NOLINT
        }
}

StringSound::StringSound(double frequency) {
        num = ceil(SAMPLING_RATE / frequency);
        buffer = new CircularBuffer(num);
        if (buffer == NULL) {
              throw std::runtime_error\
              ("StringSound Constructor: Could not allocate memory");
        }

        for (int i = 0; i < num; i++) {
            buffer->enqueue((int16_t)0);
        }

        tic_time = 0;
}

void StringSound::tic() {
        int16_t first = buffer->dequeue();
        int16_t second = buffer->peek();

        int16_t newNum = 0.994 *((first+second)/2);

        if (newNum > 0.994 *((first+second)/2)) {
            throw std::invalid_argument("tic(): Wrong value of newNum for buffer");
        }

        buffer->enqueue((sf::Int16)newNum);

        tic_time++;
}

        sf::Int16 StringSound::sample() {
        sf::Int16 sample =(sf::Int16)buffer->peek();
        return sample;
}

int StringSound::time() { // lambda expression
        int lambda = tic_time;
        auto f = [lambda] { return lambda; };
        auto lamby = f();
        return lamby;
}
```

PS5 DNA Alignment

The Assignment

The goal of this assignment was to write a program to compute the optimal sequence alignment of two DNA strings. We had to measure the similarity of two genetic sequences by their edit distance. Given two string/sequences we created a table with the dynamic programming using an N×M matrix, the Needleman-Wunsch method. We filled out the N×M matrix per the min-of-three-options formula, bottom to top, right to left which gives us the optimal edit distance in the upper-left, [0][0] cell of the matrix; Additionally, we had to traverse the matrix from top-to-bottom, left-to-right (i.e., [0][0] to [N][M]) to recover the choices you made in filling it, and thereby also recovering the actual edit sequence.

Key Concepts

In order to successfully carry out all the functions we had to make sure the table was filled with the appropriate numerical values. One of the implementations that played a key role in filling the table was within the function "alg" which consisted of:

File: dna.cpp

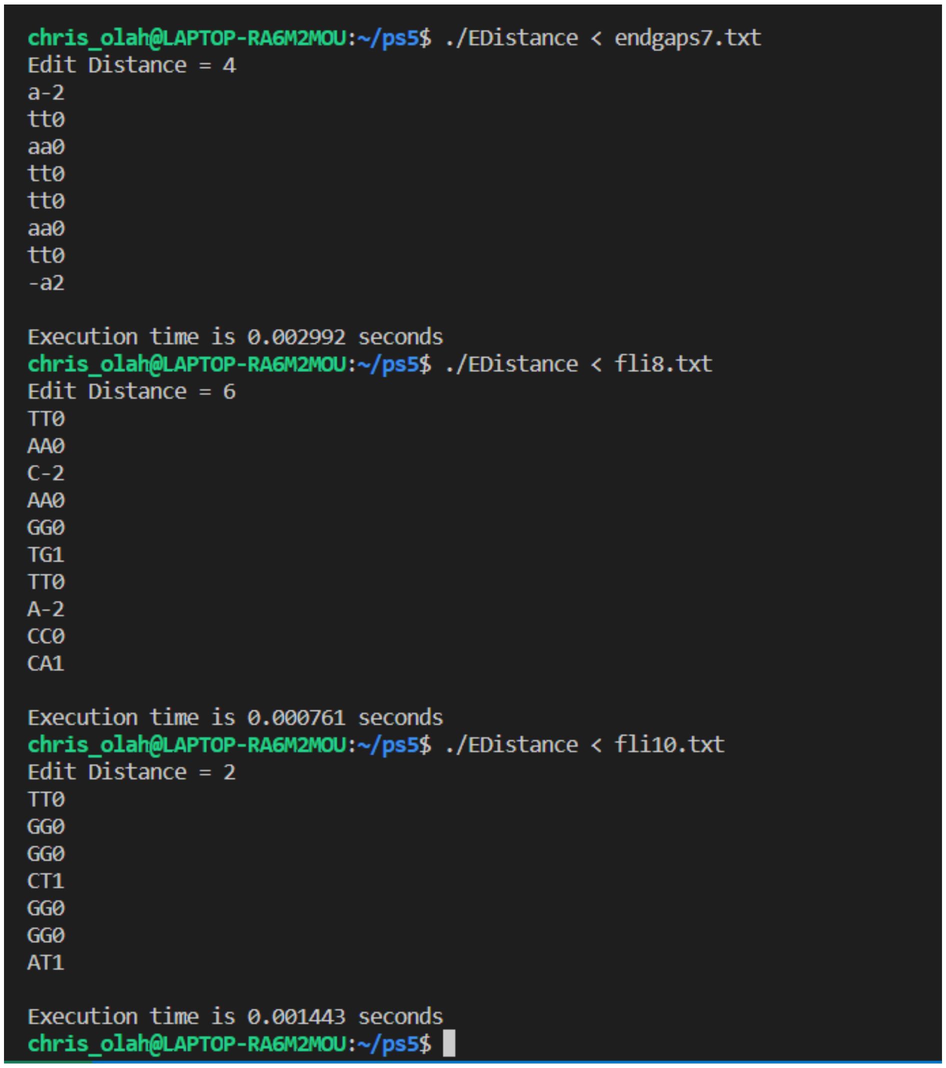
```
89: {
90: compare = penalty(dna1.at(i), dna2.at(j));
92: val = min(dnarray[i+1][j], dnarray[i][j+1], dnarray[i+1][j+1] + compare);
94: dnarray[i][j] = val;
95: }
```

This was used to carry out the pattern of the table and fill it with the correct values as it compared and found the lowest value.

What I Learned

This assignment tested our problem thinking skills as we had to find the most efficient way to carry out a pattern and it increased our knowledge of working with arrays. It was also challenging to make sure it prints out the result in the required way when printing out the sequence of the alignment function as it had to follow the correct path.

Output/Screenshot



```
chris_olah@LAPTOP-RA6M2MOU:~/ps5$ ./EDistance < endgaps7.txt
Edit Distance = 4
a-2
tt0
aa0
tt0
tt0
aa0
tt0
-a2

Execution time is 0.002992 seconds
chris_olah@LAPTOP-RA6M2MOU:~/ps5$ ./EDistance < fli8.txt
Edit Distance = 6
TT0
AA0
C-2
AA0
GG0
TG1
TT0
A-2
CC0
CA1

Execution time is 0.000761 seconds
chris_olah@LAPTOP-RA6M2MOU:~/ps5$ ./EDistance < fli10.txt
Edit Distance = 2
TT0
GG0
GG0
CT1
GG0
GG0
AT1

Execution time is 0.001443 seconds
chris_olah@LAPTOP-RA6M2MOU:~/ps5$
```

PS5 Source Code:

Makefile

```
CC= g++ -std=c++17
CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
SFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
Boost= -lboost_unit_test_framework

all: EDistance

EDistance: main.o EDistance.o
	$(CC) main.o EDistance.o -o EDistance $(SFLAGS)

main.o: main.cpp EDistance.hpp
	$(CC) -c main.cpp EDistance.hpp $(CFLAGS)

EDistance.o: EDistance.cpp EDistance.hpp
	$(CC) -c EDistance.cpp EDistance.hpp $(CFLAGS)

clean:
	rm *.o
	rm *.gch
	rm EDistance
```

main.cpp

```
// Copyright 2022 Chris Olah

#include "EDistance.hpp"

int main(int argc, char* argv[]) {
	sf::Clock clock;
	sf::Time t;

	std::string string1;
	std::string string2;

	std::cin >> string1;
	std::cin >> string2;

	EditDistance test(string1, string2);

	test.alg();
	std::cout << "Edit Distance = " << test.OptDistance() << std::endl;
	std::cout << test.Alignment() << std::endl;

	t = clock.getElapsedTime();
	std::cout << "Execution time is " << t.asSeconds() << " seconds \n";

	return 0;
}
```

EDistance.hpp

```
// Copyright 2022 Chris Olah

#ifndef _HOME_CHRIS_OLAH_PS5_EDISTANCE_HPP_
#define _HOME_CHRIS_OLAH_PS5_EDISTANCE_HPP_

#include <array>
#include <cmath>
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>

class EditDistance{
    public:
        void alg();

        ~EditDistance();

        EditDistance(std::string input1, std::string input2);

        void print();

        static int penalty(char a, char b); // returns the penalty for aligning
        // chars a and b (this will be a 0 or a 1)

        static int min(int a, int b, int c); //NOLINT
        // returns min of three arguments

        int OptDistance(); // which populates the matrix based on having the two
                            // strings, and returns the optimal distance
                            // (from the [0][0] cell of the matrix when done).

        std::string Alignment(); // which traces the matrix and
                                  // returns a string that can be
                                  // printed to display the actual alignment.
                                  // In general, this will
                                  // be a multi-line string

    private:
        int rowCount;
        int colCount;
        int** dnarray;
        std::string dna1;
        std::string dna2;
};

#endif // _HOME_CHRIS_OLAH_PS5_EDISTANCE_HPP_
```

EDistance.cpp

```
// Copyright 2022 Chris Olah

#include "EDistance.hpp"

EditDistance::EditDistance(std::string input1, std::string input2) {
    dna1 = input1;
    dna1.push_back('-');

    dna2 = input2;
    dna2.push_back('-');

    rowCount = input1.length() + 1;
    colCount = input2.length() + 1;

    dnarray = new int * [rowCount];
    for (int i = 0; i < rowCount; i++)
        dnarray[i] = new int[colCount];

    for (int i = 0; i < rowCount; i++) {
        for (int j = 0; j < colCount; j++) {
            dnarray[i][j] = 0;
        }
    }
    // setting values begining with power of two
    int exp = 0;
    for (int i = colCount - 1; i >= 0; i--) {
        dnarray[rowCount-1][i] = 2 * exp;
        exp++;
    }

    int exp1 = 0;
    for (int i = rowCount - 1; i >= 0; i--) {
        dnarray[i][colCount-1] = 2 * exp1;
        exp1++;
    }
}

int EditDistance::penalty(char a, char b) {
    if (a == b) {
        return 0;
    } else {
        return 1;
    }
}

int EditDistance:: min(int a, int b, int c) {
    a = a + 2;
    b = b + 2;
    int min = 0;
    if (a <= b && a <= c) {
        min = a;
    } else if (b <= a && b <= c) {
        min = b;
    } else if (c <= b && c <= a) {
        min = c;
    }

    return min;
}

void EditDistance::alg() {
    int val;
    int compare;

    for (int j = colCount - 2; j >= 0; j--) {
        for (int i = rowCount- 2; i >= 0; i--) {
            compare = penalty(dna1.at(i), dna2.at(j));

            val = min(dnarray[i+1][j], dnarray[i][j+1], dnarray[i+1][j+1] + compare);

            dnarray[i][j] = val;
        }
    }
}

// returning edit distance using lambda expression
int EditDistance::OptDistance() {
    int dist = dnarray[0][0];
    auto f = [dist]{return dist;};
    auto j = f();
    return j;
}
```

```

std::string EditDistance::Alignment() {
      int count;
      int i = 0;
      int j = 0;
      int distance = dnarray[0][0];
      int checker = 0;
      std::string alignment;

      while (checker != distance) {
            if (dnarray[i][j] == dnarray[i+1][j+1]+penalty\
                  (dna1.at(i), dna2.at(j))) { // DIAGONAL
                  count = dnarray[i][j]-dnarray[i+1][j+1];

                  alignment.push_back(dna1.at(i));
                  alignment.push_back(dna2.at(j));
                    alignment.push_back(count+’0’);
                    checker = checker + count;
                    alignment.push_back(’\n’);
                    if (checker != distance) {
                          i++;
                          j++;
                    } else {
                           return alignment;
                    }
              }

              if (dnarray[i][j] == dnarray[i][j+1]+2) { // Right
                    count = dnarray[i][j]-dnarray[i][j+1];
                    alignment.push_back(’-’);
                    alignment.push_back(dna2.at(j));
                    alignment.push_back(count+’0’);
                    alignment.push_back(’\n’);
                    checker = checker+count;
                     if (checker != distance) {
                           i = i + 0;
                           j++;
                     } else {
                            return alignment;
                      }
              }

```

```
        if (dnarray[i][j] == (dnarray[i+1][j])+2) { // Down
            count = dnarray[i][j]-dnarray[i+1][j];
            alignment.push_back(dna1.at(i));
            alignment.push_back('-');
            alignment.push_back(count+'0');
            alignment.push_back('\n');
            checker = checker + count;
            if (checker != distance) {
                i++;
                j = j + 0;
            } else {
                return alignment;
            }
        }
    }
    return alignment;
}

void EditDistance::print() {
    for (int i = 0; i < colCount; i++) {
        std::cout << std::setw(6) << i;
    }

    std::cout << std::endl << std::endl;

    for (int i = 0; i < rowCount; i++) {
        std::cout << i << ":";
        for (int j = 0; j < colCount; j++) {
            std::cout << std::setw(6) << dnarray[i][j];
        }
        std::cout << std::endl;
    }
}

EditDistance::~EditDistance() {
    for (int i = 0; i < rowCount; i++) {
        delete[] dnarray[i];
    }
    delete[] dnarray;
}
```

PS6 Random Writer

The Assignment

The goal of the assignment was to use the Markov Model to take in a string/text input and an order of k. Using the inputs, the program creates all the possible kgrams (of length k) and collects their frequencies along with the frequencies of possible next characters. With the data collected, and an input of L (which calculates the number of transitions) the program returns a possible output beginning with a certain kgram and the characters that may possibly follow it. We created a program that accomplishes the task above in an efficient manner.

Key Concepts

One of the most significant parts of this code is the implementation of a map within a map. As suggested in the assignment we used a map that mapped strings and another map. This secondary map mapped the individual characters.

```
std::map> mapinamap;
std::map mModel;
```

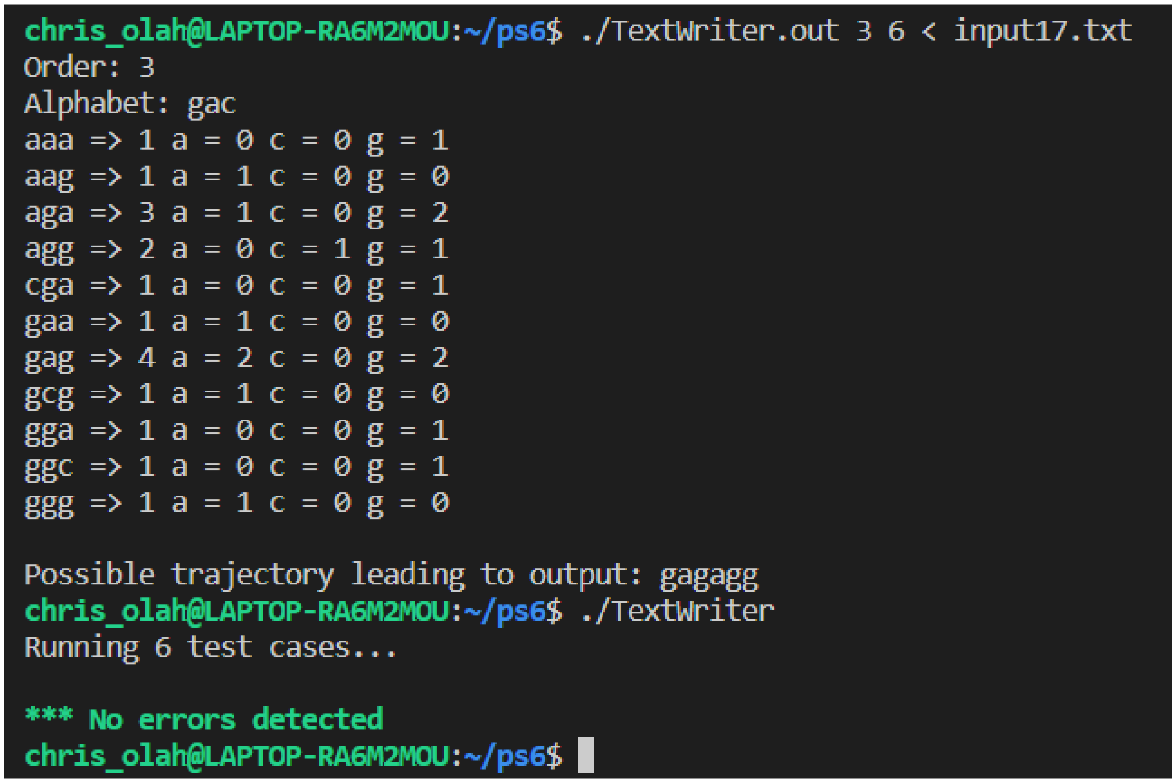
This way we were able to store the frequencies of each character and the frequencies of the kgrams in a more efficient manner and it also made it easier to print later in the program. We had other implementations that also aided in the success of the code including Krand which creates a string prob that stores all the possible values that could follow a certain kgram with respect to their frequencies. For example if a kgram "ga" has the possibility of being followed by 3 a's and 2 g's then prob would store "aaggg" within it and later when a random number is chosen within that size it will call the string with that random number as its index. File: MModel.cpp

```
for (int i = 0; i < alphabet.length(); i++) {
    total_char += mapinamap[kgram][alphabet[i]];
    for (int j = 0; j < mapinamap[kgram][alphabet[i]]; j++) {
            prob.push_back(alphabet[i]);
    }
}
result = prob[rand() % total_char];
```

What I Learned

During this process I learned more about maps and how they can be used to access information. I got more familiar with Boost testing and checking for exceptions.

Screenshot/Output



```
chris_olah@LAPTOP-RA6M2MOU:~/ps6$ ./TextWriter.out 3 6 < input17.txt
Order: 3
Alphabet: gac
aaa => 1 a = 0 c = 0 g = 1
aag => 1 a = 1 c = 0 g = 0
aga => 3 a = 1 c = 0 g = 2
agg => 2 a = 0 c = 1 g = 1
cga => 1 a = 0 c = 0 g = 1
gaa => 1 a = 1 c = 0 g = 0
gag => 4 a = 2 c = 0 g = 2
gcg => 1 a = 1 c = 0 g = 0
gga => 1 a = 0 c = 0 g = 1
ggc => 1 a = 0 c = 0 g = 1
ggg => 1 a = 1 c = 0 g = 0

Possible trajectory leading to output: gagagg
chris_olah@LAPTOP-RA6M2MOU:~/ps6$ ./TextWriter
Running 6 test cases...

*** No errors detected
chris_olah@LAPTOP-RA6M2MOU:~/ps6$
```

PS6 Source Code:

Makefile

```
CC= g++ -std=c++17
CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
SFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
Boost= -lboost_unit_test_framework

all: TextWriter TextWriter.out

TextWriter: test.o RandWriter.o
	$(CC) test.o RandWriter.o -o TextWriter $(Boost)

# Object files
TextWriter.out: TextWriter.o RandWriter.o
	$(CC) TextWriter.o RandWriter.o -o TextWriter.out $(CFLAGS)

TextWriter.o: TextWriter.cpp RandWriter.hpp
	$(CC) -c TextWriter.cpp $(CFLAGS)

test.o: test.cpp RandWriter.hpp
	$(CC) -c test.cpp RandWriter.hpp $(CFLAGS)

RandWriter.o: RandWriter.cpp RandWriter.hpp
	$(CC) -c RandWriter.cpp $(CFLAGS)

# Cleanup object files
clean:
	rm *.o
	rm TextWriter
```

TextWriter.cpp

```
// Copyright 2022 Chris Olah
#include <string>
#include <stdexcept>
#include <exception>
#include <fstream>
#include "RandWriter.hpp"

int main(int argc, char* argv[]) {
	int orderk = atof(argv[1]);
	int L = atof(argv[2]);

	std::string text;
	std::string line;
	std::string kgram;

	while (getline(std::cin, line)) {
		text += line;
	}

	for ( int i = 0 ; i < orderk; i++ ) {
		kgram.push_back(text[i]);
	}

	auto lambda = [](int n) {return n;};
	RandWriter randwriter(text, lambda(orderk));

	std::cout << randwriter << "\n";
	std::cout << "Possible trajectory leading to output: " << \
	randwriter.generate(kgram, L) << "\n";

	return 0;
}
```

RandWriter.hpp

```
// Copyright 2022 Chris Olah

#ifndef _HOME_CHRIS_OLAH_PS6_RANDWRITER_HPP_
#define _HOME_CHRIS_OLAH_PS6_RANDWRITER_HPP_

#include <iostream>
#include <string>
#include <map>

class RandWriter {
	public:
		RandWriter(std::string text, int k);
		int orderK() const;
		int freq(std::string kgram);
		int freq(std::string kgram, char c) const;
		char kRand(std::string kgram);
		std::string generate(std::string kgram, int L);
		friend std::ostream& operator<<(std::ostream& out, const RandWriter& m);

	private:
		std::map<std::string, std::map<char, int>> mapinamap;
		std::map<std::string, int> randwriter;
		std::string input_text;
		std::string alphabet;
		int orderk;
};

#endif // _HOME_CHRIS_OLAH_PS6_RANDWRITER_HPP_
```

RandWriter.cpp

```
// Copyright 2022 Chris Olah

#include "RandWriter.hpp"
#include <string>
#include <map>
#include <algorithm>

// Constructor builds markov model
RandWriter::RandWriter(std::string text, int k) {
	input_text = text;
	std::string input_temp = text;
	orderk = k;
	alphabet = text[0]; // stores the different alphabets/symbols

	for (int i = 0; (unsigned) i < text.length(); i++) {
		if (!(alphabet.find(text[i]) != std::string::npos)) {
			alphabet.push_back(text[i]);
		}

		if ((unsigned) i + k > text.length()-1) {
			input_temp += text;
			std::string temp2;

			for (int j = 0; j < k; j++) {
				temp2.push_back(input_temp[i + j]);
			}

			// makes sure the same values aren''t being stored
			if (!(randwriter.find(temp2) != randwriter.end())) {
				randwriter[temp2] = 0;
			}

			for (std::map<std::string, int>::iterator it = randwriter.begin();
				it != randwriter.end(); ++it) {
					if (temp2 == it->first) {
						it->second += 1;
					}
				}
		}

```

```
            if ((unsigned) i + k <= text.length() - 1) {
            std::string temp;

            for (int j = 0; j < k; j++) {
                temp.push_back(text[i + j]);
            }

            if (!(randwriter.find(temp) != randwriter.end())) {
                randwriter[temp] = 0;
            }

            for (std::map<std::string, int>::iterator it = randwriter.begin();
                it != randwriter.end(); ++it) {
                    if (temp == it->first) {
                        it->second += 1;
                    }
                }
            }
    }

    std::map<char, int> char_prob;
    for (auto& t : randwriter) {
            for (int i = 0; (unsigned) i < alphabet.length(); i++) {
                char_prob[alphabet[i]] = freq(t.first, alphabet[i]);
            }
            mapinamap[t.first] = char_prob;
    }
}

int RandWriter::orderK() const {
    return orderk;
}

int RandWriter::freq(std::string kgram) {
    if (kgram.length() != (unsigned) orderk) {
            throw std::invalid_argument("kgram must be of length k");
    }

    if (!(randwriter.find(kgram) != randwriter.end())) {
            throw std::runtime_error("kgram doesn't exist");
    }

    return randwriter[kgram];
}

int RandWriter::freq(std::string kgram, char c) const {
    int counter = 0;
    int check;

    if (!(randwriter.find(kgram) != randwriter.end())) {
        throw std::runtime_error("kgram doesn’t exist");
    }

    for (int i = 0; (unsigned) i < input_text.length(); i++) {
        check = i + orderk;

        if ((unsigned) check > input_text.length() - 1) {
            std::string text_temp = input_text;
            text_temp += input_text;

            std::string temp2;
            char tempc2;

            for (int j = 0; j < orderk; j++) {
                temp2.push_back(text_temp[i + j]);
            }

            tempc2 = text_temp[i + orderk];

            if (kgram == temp2 && c == tempc2) {
                counter++;
            }
        }

        if ((unsigned) check <= input_text.length() - 1) {
            std::string temp;
            char tempc;

            for (int j = 0; j < orderk; j++) {
                temp.push_back(input_text[i + j]);
            }

            tempc = input_text[i + orderk];

```

```
                if (kgram == temp && c == tempc) {
                    counter++;
                }
            }
    }

    return counter;
}

char RandWriter::kRand(std::string kgram) {
    int total_char = 0;
    std::string prob;
    char result;

    if (kgram.length() != (unsigned) orderk) {
        throw std::invalid_argument("kgram must be of length k");
    }

    // prob stores all the frequencies of characters and when
    // random number is chosen it selects a character from prob
    for (int i = 0; (unsigned) i < alphabet.length(); i++) {
        total_char += mapinamap[kgram][alphabet[i]];

        for (int j = 0; j < mapinamap[kgram][alphabet[i]]; j++) {
            prob.push_back(alphabet[i]);
        }
    }

    result = prob[rand() % total_char];

    return result;
}

std::string RandWriter::generate(std::string kgram, int L) {
    std::string generate = kgram;
    std::string temp = kgram;
    std::string next_gram = kgram;
    int transitions = L - orderk; // L-k transitions
    int track = 0;

    for (int i = 0; i < transitions; i++) {
        generate.push_back(kRand(next_gram));

        if ((unsigned) i + orderk <= generate.length() - 1) {
            next_gram = generate.substr(track + 1, generate.length());
        }
        track++;
    }

    return generate;
}

std::ostream& operator<<(std::ostream& out, const RandWriter& m) {
    out << "Order: " << m.orderk << "\n";
    out << "Alphabet: " << m.alphabet << "\n";

    for (auto it : m.mapinamap) {
        out << it.first << " => " << m.randwriter.find(it.first)->second;

        std::map<char, int>& internal_map = it.second;

        for (auto it2 : internal_map) {
            out << " " << it2.first << " = " << it2.second;
        }

        out << "\n";
    }

    return out;
}
```

test.cpp

```
// Copyright 2022 Chris Olah

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

BOOST_AUTO_TEST_CASE(Constructor) {
    BOOST_REQUIRE_NO_THROW(RandWriter("gagggagaggcgagaaa", 2));
    BOOST_REQUIRE_NO_THROW(RandWriter("gagggagaggcgagaaa", 5));
}

// Checks the freq(string kgram)
BOOST_AUTO_TEST_CASE(freq) {
    RandWriter test("gagggagaggcgagaaa", 2);

    // shouldn’t fail
    BOOST_REQUIRE_NO_THROW(test.freq("ga"));

    // These should fail.
    BOOST_REQUIRE_THROW(test.freq("gac"), std::invalid_argument);
    BOOST_REQUIRE_THROW(test.freq("da"), std::runtime_error);

    BOOST_REQUIRE(test.freq("ga") == 5);
    BOOST_REQUIRE(test.freq("ag") == 5);
    BOOST_REQUIRE(test.freq("gg") == 3);
}

// Checks the freq(string kgram, char c) method
BOOST_AUTO_TEST_CASE(Freq) {
    RandWriter test("gagggagaggcgagaaa", 2);

    // no exception thrown
    BOOST_REQUIRE_NO_THROW(test.freq("ga", ’a’));

    // should fail
    BOOST_REQUIRE_THROW(test.freq("gac", ’c’), std::invalid_argument);
    BOOST_REQUIRE_THROW(test.freq("da", ’d’), std::runtime_error);

    BOOST_REQUIRE(test.freq("ga", ’a’) == 1);
    BOOST_REQUIRE(test.freq("ag", 'g') == 2);
    BOOST_REQUIRE(test.freq("gg", 'c') == 1);
}

// Checks the kRand() method
BOOST_AUTO_TEST_CASE(kRand) {
    RandWriter test("gagggagaggcgagaaa", 2);

    // shouldn't fail.
    BOOST_REQUIRE_NO_THROW(test.kRand("ag"));

    // These should fail.
    BOOST_REQUIRE_THROW(test.kRand("agc"), std::invalid_argument);
    BOOST_REQUIRE_THROW(test.kRand("agbda"), std::invalid_argument);
}

// test korder
BOOST_AUTO_TEST_CASE(kOrder) {
    RandWriter test("gagggagaggcgagaaa", 2);
    BOOST_REQUIRE(test.orderK() == 2);

    RandWriter test2("gagggagaggcgagaaa", 5);
    BOOST_REQUIRE(test2.orderK() == 5);
}

// test generate
BOOST_AUTO_TEST_CASE(textgenerator) {
    RandWriter test("gagggagaggcgagaaa", 2);
    BOOST_REQUIRE((test.generate("ag", 7)).length() == (2+5));
    BOOST_REQUIRE((test.generate("ag", 10)).length() == (2+8));
    BOOST_REQUIRE((test.generate("gg", 5)).length() == (2+3));
}
```

PS7 Kronos Time Parsing

The Assignment

The goal of PS7 was to use the boost::regex library to replicate the Kronos InTouch Linux based touch-screen time clocks. When the InTouch device boots up, the program produces a first logging message that will indicate the server has started ((log.c..166) server started). Then, the completion of the boot up sequence can be seen via the log entry for the device Softkeys being displayed on the UI: (oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080). Each startup is gone through and the program produces whether the bootup completed successfully or not along with how much time was required to boot up. This information is produced in an output log file.

Key Concepts

The most significant part of this assignment was the use of regular expressions created to parse through the intouch log files. The two significant regex expressions in my program are listed below in the main.cpp file:

36) boost::regex start("[(]log[.]c[.]166[)] server started")

This regex only boots startups that have the same exact string. It checksfor the exact expression (log.c.166) server started

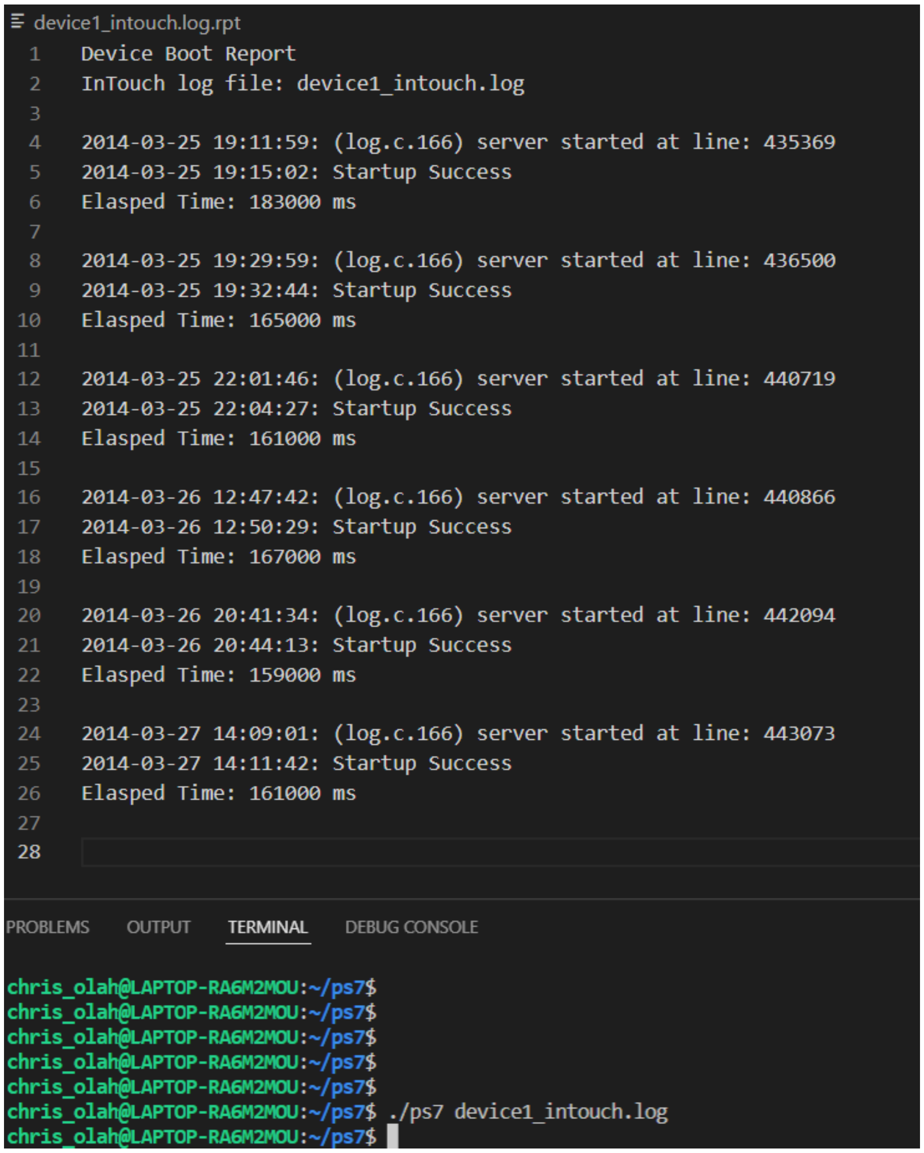
37) boost::regex end("oejs[.]AbstractConnector:Started SelectChannelConnector");

This regex only boots completions that have the exact same string. This checks for the expression oejs.AbstractConnector:Started SelectChannelConnector

What I Learned

Programming this assignment, I strengthened my knowledge of using regular expressions to output properly formatted output data. I also further learned about outputting data into a log file. Another thing, I solidified my knowledge of using boost tests to assure the program was outputting correct via the stdin_boost.cpp file.

Output/Screenshot



```
Device Boot Report
InTouch log file: device1_intouch.log

2014-03-25 19:11:59: (log.c.166) server started at line: 435369
2014-03-25 19:15:02: Startup Success
Elasped Time: 183000 ms

2014-03-25 19:29:59: (log.c.166) server started at line: 436500
2014-03-25 19:32:44: Startup Success
Elasped Time: 165000 ms

2014-03-25 22:01:46: (log.c.166) server started at line: 440719
2014-03-25 22:04:27: Startup Success
Elasped Time: 161000 ms

2014-03-26 12:47:42: (log.c.166) server started at line: 440866
2014-03-26 12:50:29: Startup Success
Elasped Time: 167000 ms

2014-03-26 20:41:34: (log.c.166) server started at line: 442094
2014-03-26 20:44:13: Startup Success
Elasped Time: 159000 ms

2014-03-27 14:09:01: (log.c.166) server started at line: 443073
2014-03-27 14:11:42: Startup Success
Elasped Time: 161000 ms

```

```
chris_olah@LAPTOP-RA6M2MOU:~/ps7$
chris_olah@LAPTOP-RA6M2MOU:~/ps7$
chris_olah@LAPTOP-RA6M2MOU:~/ps7$
chris_olah@LAPTOP-RA6M2MOU:~/ps7$
chris_olah@LAPTOP-RA6M2MOU:~/ps7$
chris_olah@LAPTOP-RA6M2MOU:~/ps7$ ./ps7 device1_intouch.log
chris_olah@LAPTOP-RA6M2MOU:~/ps7$
```

PS7 Source Code:

Makefile

```
CC = g++
Debug = -g
CFLAGS = -c -Wall -Werror -std=c++11
OBJECTS = main.o
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-audio -lsfml-system
REGEX = -lboost_regex -lboost_date_time
TEST = -lboost_unit_test_framework
all:
	make ps7
ps7: $(OBJECTS)
	g++ -Wall $(OBJECTS) -o ps7 $(LFLAGS) $(REGEX)
main.o: main.cpp
	$(CC) $(CFLAGS) -c main.cpp -o main.o $(LFLAGS) $(REGEX)
clean:
	rm $(OBJECTS) ps7
```

main.cpp

```cpp
// Copyright 2022 Chris Olah
#include <iostream>
#include <cstdint>
#include <vector>
#include <memory>
#include <exception>
#include <stdexcept>
#include <fstream>
#include <string>
#include <boost/regex.hpp>
#include "boost/date_time/gregorian/gregorian.hpp"
#include "boost/date_time/posix_time/posix_time.hpp"

using boost::gregorian::date;
using boost::gregorian::from_simple_string;
using boost::gregorian::date_period;
using boost::gregorian::date_duration;
using boost::posix_time::ptime;
using boost::posix_time::time_duration;

int main(int argc, char* argv[]) {
    auto lambda = [](std::string n) {return n;};
    std::ifstream kronos(argv[1]);
    std::string input_file = argv[1];

    int line = 0;
    int started = 0;

    date d1;
    date d2;
    ptime t1;
    ptime t2;

    std::ofstream output_file(input_file + ".rpt", std::ofstream::out);
    std::string current_line;
    boost::regex start("[(]log[.]c[.]166[)] server started");
    boost::regex end("oejs[.]AbstractConnector:Started SelectChannelConnector");

    if (kronos.is_open() && output_file.is_open()) {
        output_file << "Device Boot Report\n"
        << "InTouch log file: " << input_file << "\n\n";

        while (getline(kronos, current_line)) {
            ++line;
            if (boost::regex_search(current_line, start)) {
                if (started == 1) {
                    output_file << "Startup failed \n\n";
                    started = 0;
                }
                output_file << current_line << "at line: " << line << "\n";
                started = 1;

                std::string temp_date = current_line.substr(0, 10);
                std::string temp_time = current_line.substr(11, 8);

                d1 = from_simple_string(lambda(temp_date));
            ptime temp(d1, time_duration(stoi(temp_time.substr(0, 2)), stoi(temp_time.substr(3, 2)), stoi(temp_time.substr(6, 2)), 0)); // NOLINT
            t1 = temp;
        } else if (boost::regex_search(current_line, end)) {
            std::string temp_date = current_line.substr(0, 10);
            std::string temp_time = current_line.substr(11, 8);

            d2 = from_simple_string(temp_date);
            ptime temp(d2, time_duration(stoi(temp_time.substr(0, 2)), stoi(temp_time.substr(3, 2)), stoi(temp_time.substr(6, 2)), 0)); // NOLINT
            t2 = temp;
            std::string date_time = current_line.substr(0, 19);
            time_duration td = t2-t1;
            output_file << date_time << ": " << "Startup Success \n"
            << "Elasped Time: " << td.total_milliseconds() << " ms\n\n";
            started = 0;
        }
    }
    }
    if (started == 1) {
        output_file << "Startup failed \n\n";
        started = 0;
    }
    kronos.close();
    output_file.close();

```

```
	return 0;
}
```

datetime.cpp

```
// date and time sample code
// Copyright (C) 2015 Fred Martin
// Tue Apr 21 17:37:46 2015

// compile with
// g++ datetime.cpp -lboost_date_time
// Y. Rykalova 4/12/2021

// http://www.boost.org/doc/libs/1_58_0/doc/html/date_time/gregorian.html
// http://www.boost.org/doc/libs/1_58_0/doc/html/date_time/posix_time.html

#include <iostream>
#include <string>
#include "boost/date_time/gregorian/gregorian.hpp"
#include "boost/date_time/posix_time/posix_time.hpp"

using std::cout;
using std::cin;
using std::endl;
using std::string;

using boost::gregorian::date;
using boost::gregorian::from_simple_string;
using boost::gregorian::date_period;
using boost::gregorian::date_duration;

using boost::posix_time::ptime;
using boost::posix_time::time_duration;

int main() {
    // Gregorian date stuff
    string s("2015-01-01");
    date d1(from_simple_string(s));
    date d2(2015, boost::gregorian::Apr, 21);

    date_period dp(d1, d2); // d2 minus d1

    date_duration dd = dp.length();

    cout << "duration in days " << dd.days() << endl;

    // Posix date stuff
    ptime t1(d1, time_duration(0, 0, 0, 0)); // hours, min, secs, nanosecs
    ptime t2(d2, time_duration(0, 0, 0, 0));

    time_duration td = t2 - t1;

    cout << "duration in hours " << td.hours() << endl;
    cout << "duration in ms " << td.total_milliseconds() << endl;
}
```

stdin_boost.cpp

```
// compile with
// g++ stdin_boost.cpp -lboost_regex

// regex_match example
#include <iostream>
#include <string>
#include <boost/regex.hpp>

using namespace std;
using namespace boost;

int main () {
	string s, rs;
	regex e;

	// see http://www.boost.org/doc/libs/1_55_0/boost/regex/v4/error_type.hpp
	cout << "Here are some helpful error codes you may encounter\n";
	cout << "while constructing your regex\n";
	cout << "error_bad_pattern " << regex_constants::error_bad_pattern << endl;
	cout << "error_collate " << regex_constants::error_collate << endl;
	cout << "error_ctype " << regex_constants::error_ctype << endl;
	cout << "error_escape " << regex_constants::error_escape << endl;
	cout << "error_backref " << regex_constants::error_backref << endl;
	cout << "error_brack " << regex_constants::error_brack << endl;
	cout << "error_paren " << regex_constants::error_paren << endl;
	cout << "error_brace " << regex_constants::error_brace << endl;
	cout << "error_badbrace " << regex_constants::error_badbrace << endl;

	cout << endl;

	cout << "Enter regex > ";
	getline (cin, rs);

	try {
		e = regex (rs);
	} catch (regex_error& exc) {
		cout << "Regex constructor failed with code " << exc.code() << endl;
		exit(1);
	}

```

```
    cout << "Enter line > ";

    while (getline(cin, s)) {

        cout << endl;

        if (regex_match (s,e))
            cout << "string object \"" << s << "\" matched\n\n";

        if ( regex_match ( s.begin(), s.end(), e ) )
            cout << "range on \"" << s << "\" matched\n\n";

        smatch sm; // same as match_results<string::const_iterator> sm;
        regex_match (s,sm,e);
        cout << "string object \"" << s << "\" with " << sm.size() << " matches\n\n";
        // uses constant iterators so requires -std=gnu++0x
        // regex_match ( s.cbegin(), s.cend(), sm, e);
        // cout << "range on \"" << s << "\" with " << sm.size() << " matches\n";

        if (sm.size() > 0) {
        cout << "the matches were: ";
        for (unsigned i=0; i<sm.size(); ++i) {
            cout << "[" << sm[i] << "] " << endl;
        }
    }

    cout << endl << endl;

    cout << "Enter line > ";
    }


    return 0;
}
```