Visoth Andy Pen
COMP IV: Project Portfolio
April 26, 2021
Time to Complete: 4 hours

Contents:

PS0: Hello World with SFML

Assignment

This assignment involved installing virtual box with ubuntu and getting used to the environment. It involved building a SFML hello world code that checked if the code was running properly and displays a sprite image.

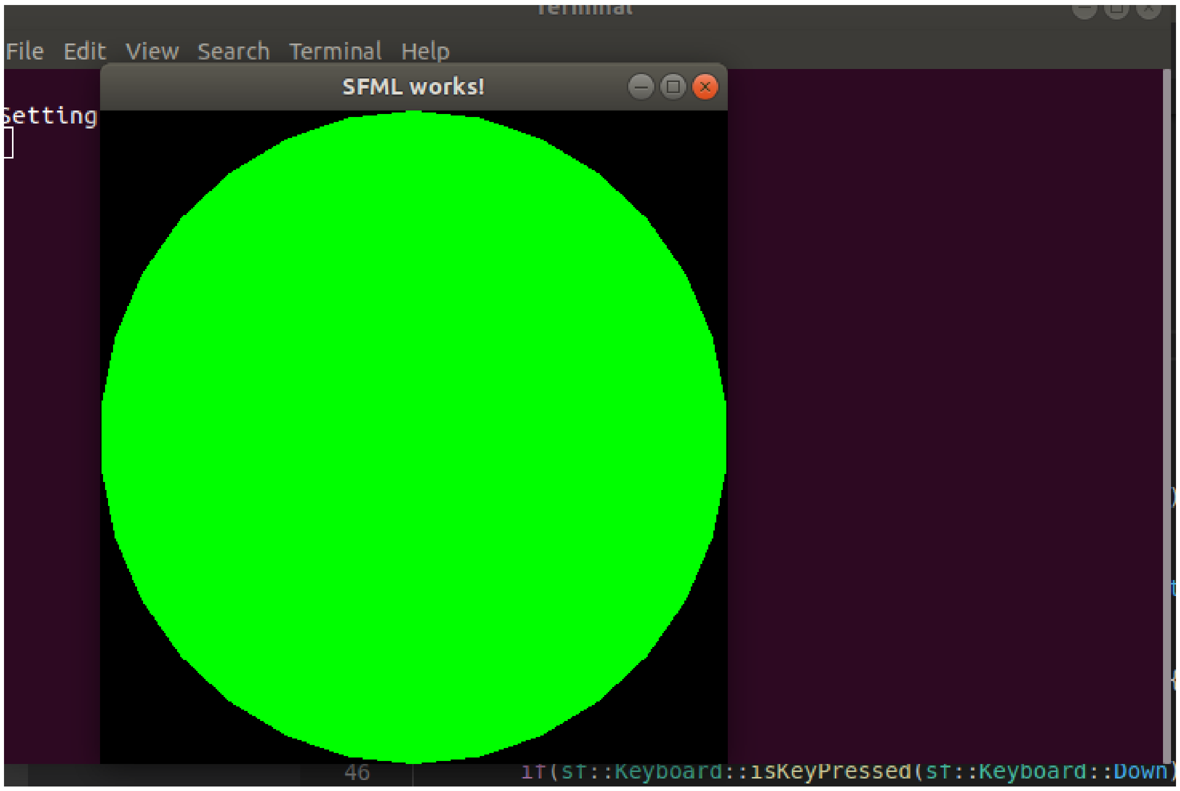
Key Concepts

This assignment involved using basic programming such as variables and loops to display the sprite in the window. This assignment tested out SFML's different featuressuch as images, sprites, texture, and keyboard functions. I had to load an image into a texture and use the sprite to set the texture. The SFML's sprite classwas used to display a moving image in a window. The assignment required to add functions to the sprite so I added the arrow keyboard functions to move the sprite up, down, left, or right depending which arrow key was pressed.

What I Learned

In this assignment, I learned how to use SFML basic features such as how to display images in an SFML window and how to move the sprite based on keyboard functions. I also learned how to setup a Linux environment using the virtual box which allows me to have Ubuntu on my windows system. I already knew how to use Linux's basic features because I had to use the system for previous computing classes, but I didn't know how to install the virtual machines.

Screenshot:



```
Setting
```

```
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down)
```

Makefile:

```
CXX                 := g++
CXX_FLAGS := -Wall -Wextra -std=c++17 -ggdb

BIN               := bin
SRC               := src
INCLUDE   := include
LIB               := lib

LIBRARIES := -lsfml-graphics -lsfml-window -lsfml-system
EXECUTABLE      := main


all: $(BIN)/$(EXECUTABLE)

run: clean all
        clear
        ./$(BIN)/$(EXECUTABLE)

$(BIN)/$(EXECUTABLE): $(SRC)/*.cpp
        $(CXX) $(CXX_FLAGS) -I$(INCLUDE) -L$(LIB) $^ -o $@ $(LIBRARIES)

clean:
        -rm $(BIN)/*

```

Main.cpp

```cpp
/*

Name: Andy Pen
Date: 1/31/2021
 */
#include <SFML/Graphics.hpp>
//#include <SFML/Window.hpp>
#include <SFML/Window/Keyboard.hpp>
#include <iostream>

 int main()
 {
     sf::RenderWindow window(sf::VideoMode(800, 800), "SFML works!");
     sf::CircleShape shape(100.f);
     shape.setFillColor(sf::Color::Green);
     sf::CircleShape sprite(200.f);
     sprite.setPosition(206.0f, 206.0f);

     sf::Texture spriteTexture;

     if(!spriteTexture.loadFromFile("sprite.png")){
         std::cout << "Unable to open file" << std::endl;
     }
     sprite.setTexture(&spriteTexture);

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



         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)){
                sprite.move(sf::Vector2f(-0.1f, 0.0f));
         }
         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
                 sprite.move(sf::Vector2f(0.1f, 0.0f));
         }
         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up)){
                 sprite.move(sf::Vector2f(0.0f, -0.1f));
         }
         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down)){
                 sprite.move(sf::Vector2f(0.0f, 0.1f));
         }

         if(sf::Keyboard::isKeyPressed(sf::Keyboard::R)){
                 sprite.rotate(1.0f);
         }


```

```

        window.clear();
        window.draw(shape);
       window.draw(sprite);
        window.display();
    }

    return 0;
}

```

PS1a: Linear Feedback Shift Register

Assignment

This assignment uses the concept of the linear feedback shift register that produce pseudo-random bits that will be used in the next assignment to encrypt and decrypt images. I had to implement 16 bits Fibonacci LFSR class which took a constructor given an initial seed and tap, which then simulated and generated steps to produce the bits. This assignment also used Boost test framework to implement unit tests.

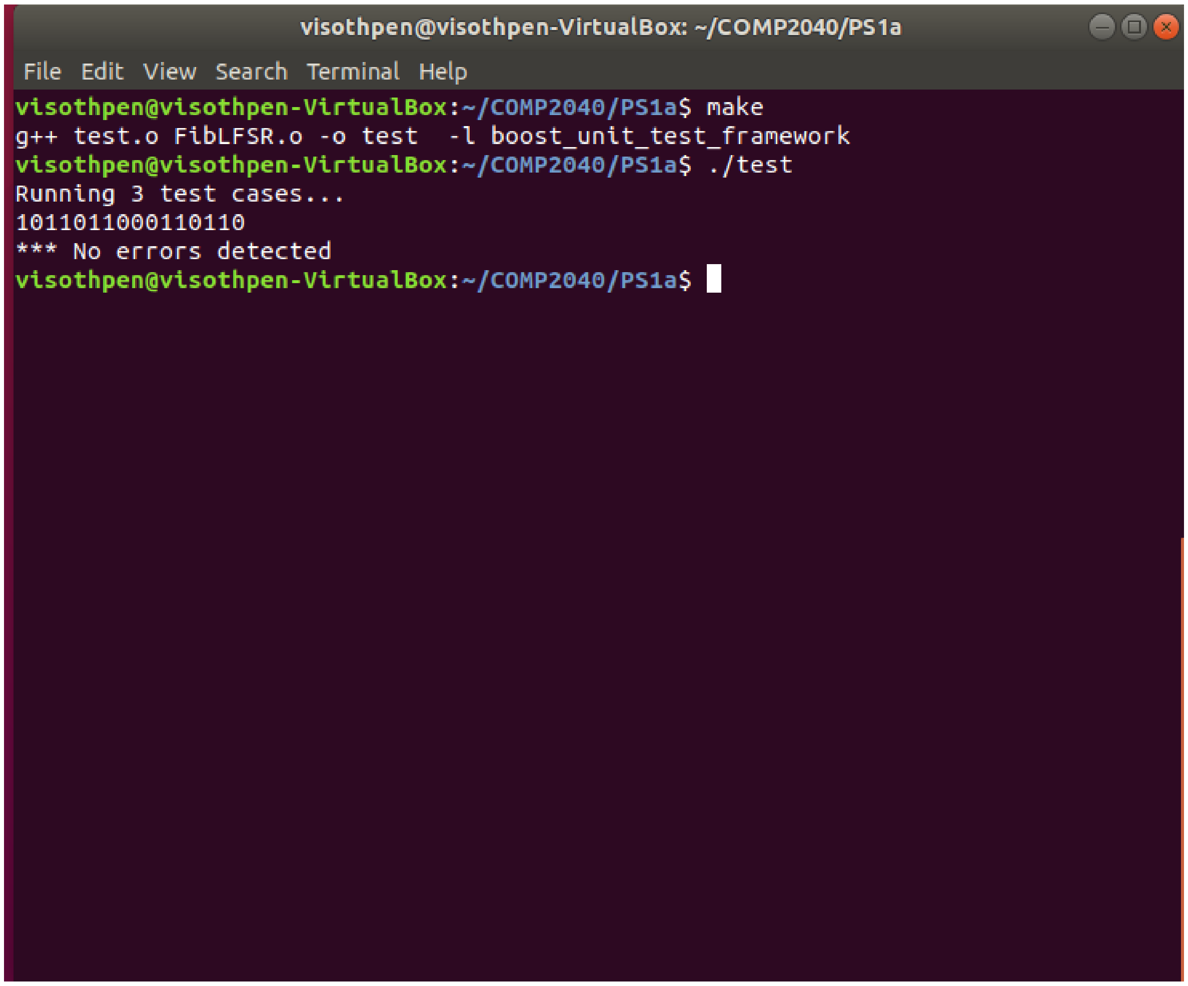
Key Concepts

This assignment used the concept of 16 bit Fibonnaci LFSR to perform steps that would XOR the bit and shifting it left by feeding the string representing the register into an object which would then XOR the left most bit with the tap position. Basically, a LFSR was created using the initial seed and tap to simulate one step that would return a new bit as 0 or 1 and keep generating those steps. Also, the Boost test framework was used to test the implementations with unit testing.

What I Learned

In this assignment, I learned the concept of linear feedback shift registers and how to use Boost test frameworks for unit testing. The unit testing helped me figure out whether or not I did the implementation correctly for the functions. I also learned more on Makefiles in which I only knew the basics of, but this assignment required to create a Makefile from scratch to compile the program.

Screenshot:


visothpen@visothpen-VirtualBox: ~/COMP2040/PS1a
File Edit View Search Terminal Help

```
visothpen@visothpen-VirtualBox:~/COMP2040/PS1a$ make
g++ test.o FibLFSR.o -o test  -l boost_unit_test_framework
visothpen@visothpen-VirtualBox:~/COMP2040/PS1a$ ./test
Running 3 test cases...
1011011000110110
*** No errors detected
visothpen@visothpen-VirtualBox:~/COMP2040/PS1a$ 
```

Makefile:

```
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
OBJ = test.o FibLFSR.o
DEPS = FibLFSR.cpp test.cpp FibLFSR.h
LIBS = -l boost_unit_test_framework
EXE = test

all: $(OBJ)
	$(CC) $(OBJ) -o $(EXE) $(LIBS)

 %.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -o $@ $<

 clean:
	rm $(OBJ) $(EXE)

```

Test.cpp

```cpp
// (C) Copyright Andy Pen, 2021.
#include <iostream>
#include <string>
#include "FibLFSR.h"

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

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);
}

BOOST_AUTO_TEST_CASE(secondTestCase) {
  FibLFSR l("1011011000110110");
  // test ostream operator
  std::cout << l;
}


BOOST_AUTO_TEST_CASE(thirdTestCase) {
  // exception handling
  try {
    FibLFSR l("12135431");
    BOOST_ERROR("Invalid input");
  } catch (std::invalid_argument) {}
}

```

FibLFSR.h

```cpp
// (C) Copyright Andy Pen, 2021.
#ifndef _HOME_VISOTHPEN_COMP2040_PSXA_FIBLFSR_H_
#define _HOME_VISOTHPEN_COMP2040_PSXA_FIBLFSR_H_
#include <iostream>
#include <string>
#include <bitset>

class FibLFSR {
 public:
     FibLFSR(const std::string& seed);  //NOLINT
     //  the given initial seed and tap
     int step();  //  simulate one step and return the
     //  new bit as 0 or 1
     int generate(int k);  // simulate k steps and return
     // k-bit integer
     friend std::ostream& operator <<(std::ostream& lhs, const FibLFSR&
rhs);
  private:
     std::bitset<16> value;
 };
 #endif  // _HOME_VISOTHPEN_COMP2040_PSXA_FIBLFSR_H_
```

FibLFSR.cpp

```cpp
// (C) Copyright Andy Pen, 2021.
#include "FibLFSR.h"
#include <stdlib.h>
#include <stdio.h>


FibLFSR::FibLFSR(const std::string &seed) : value{seed} {}

int FibLFSR::step() {  // simulate one step and return the
                                  // new bit as 0 or 1
     int newBit = value[15] ^ value[13] ^ value[12] ^ value[10];
     // shift value
     value = value << 1;
     value[0] = newBit;
     return newBit;
 }
 int FibLFSR::generate(int k) {  // simulate k steps and return
                                  // k-bit integer
     int retVal = 0;
     int newVal = 0;
     for (int i = 0; i < k; i++) {
         newVal = step();
         retVal = (retVal *2) + newVal;
     }
     return retVal;
 }
 std::ostream& operator <<(std::ostream& out, const FibLFSR& rhs) {
     out << rhs.value;
     return out;
 }

```

PS1b: LFSR - PhotoMagic

Assignment

This assignment uses the implementation of the previous assignment to transform an image and display the encrypted copy of the image using SFML. The program reads in three arguments from the command line; the source image filename, output filename, and the FibLFSR seed. The image is then loaded and displayed using SFML in which the image is encoded/decoded using the FibLFSR class.

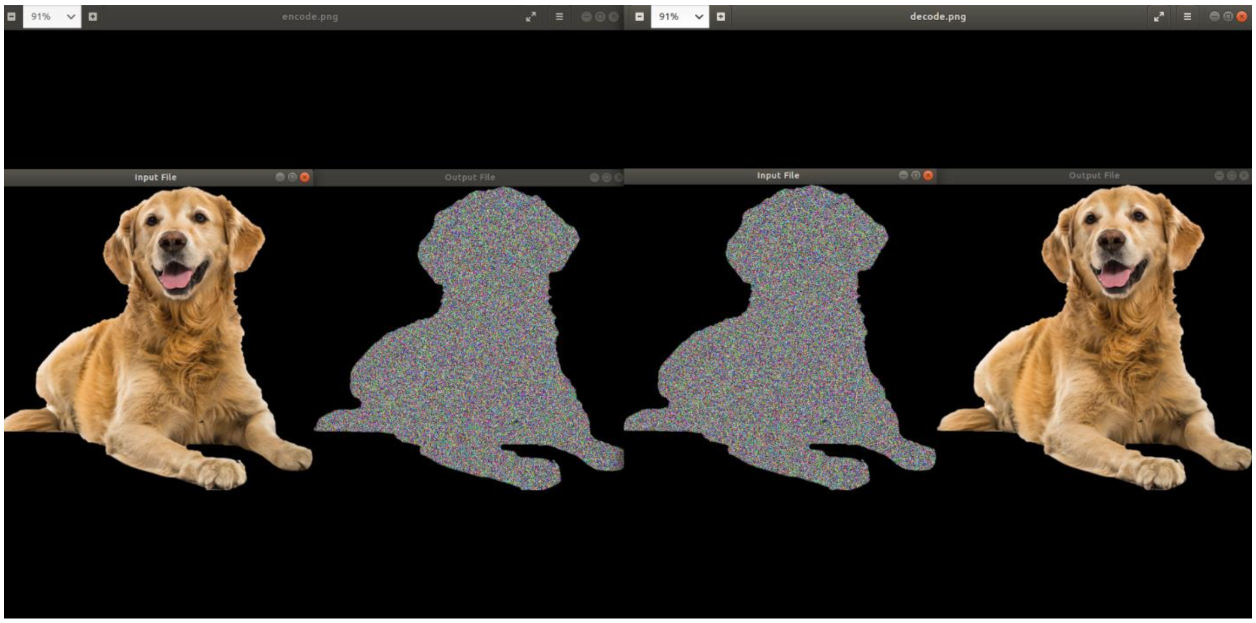
Key Concepts

This assignment used the FibLFSR class and SFML features to load an image and encode/decode the image, and then display the image after saving it to the disk. A transform function was implemented using FIbLFSR to change the pixels of the images which would encrypt/decypt it.

What I Learned

Since this assignment was derived from the previous, there wasn't much more learned about the FibLFSR class, but I did learn about image's and their pixels. The pixels had RGB colors in the form (x,y) in which the FibLFSR class was used to XOR each component to create the new pixels.

Screenshot:


91%
encode.png
91%
decode.png
Input File
Output File
Input File
Output File

Makefile:

```
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
OBJ = PhotoMagic.o FibLFSR.o
DEPS = FibLFSR.cpp PhotoMagic.cpp FibLFSR.h
LIBS = -lsfml-graphics -lsfml-window -lsfml-system
EXE = PhotoMagic

all: $(OBJ)
  $(CC) $(OBJ) -o $(EXE) $(LIBS)

 %.o: %.cpp $(DEPS)
         $(CC) $(CFLAGS) -o $@ $<

 clean:
         rm $(OBJ) $(EXE)

```

PhotoMagic.cpp

```cpp
/*
Andy Pen
COMP2040
Dr.Rykalova
2/15/2021
*/
#include "FibLFSR.h"
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
using namespace std;

void transform(sf::Image& image, FibLFSR* fib);

// display an encrypted copy of the picture, using the LFSR
// to do the encryption
int main(int argc, char* argv[]){

    //read in filenames and string bit
    string str = argv[3];
    string str2 = argv[1];
    string str3 = argv[2];

    sf::Image image;
    sf::Image image2;
    if(!image.loadFromFile(str2)){
        return -1;
    }
    if(!image2.loadFromFile(str2)){
        return -1;
    }
    //create Fib pointer to read in third argument of string bits
    FibLFSR *ptr = new FibLFSR(str);
    //call transform function to encrypt source file
    transform(image2, ptr);
    delete ptr;

    //save transformed image to ouput-file
    image2.saveToFile(str3);

    sf::Vector2u size = image.getSize();
    sf::Vector2u size2 = image2.getSize();

    sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Input File");
    sf::RenderWindow window2(sf::VideoMode(size2.x, size2.y), "Output File");
    sf::Texture texture;
    texture.loadFromImage(image);

    sf::Texture texture2;
    texture2.loadFromImage(image2);

    sf::Sprite sprite;
    sprite.setTexture(texture);

    sf::Sprite sprite2;
    sprite2.setTexture(texture2);

while (window1.isOpen() && window2.isOpen()) {
    sf::Event event;
    while (window1.pollEvent(event)) {
        if (event.type == sf::Event::Closed)
            window1.close();
    }
    while (window2.pollEvent(event)) {
        if (event.type == sf::Event::Closed)
            window2.close();
    }
    window1.clear();
    window1.draw(sprite);
    window1.display();
    window2.clear();
    window2.draw(sprite2);
    window2.display();
}


    return 0;
}

void transform(sf::Image& image, FibLFSR* fib){ // transforms image using FibLFSR

    sf::Color p;
    sf::Vector2u size = image.getSize(); //figure out how big image is using getSize
    for(unsigned int x = 0; x < size.x; x++){
        for(unsigned int y = 0; y < size.y; y++){
            p = image.getPixel(x,y);
            p.r = p.r ^ fib->generate(8);
            p.g = p.g ^ fib->generate(8);
            p.b = p.b ^ fib->generate(8);
            image.setPixel(x,y,p);

        }
    }
}

```

FibLFSR.h

```cpp
/*
Andy Pen
Dr.Rykalova
Due 2/8/21
*/
#include <iostream>
#include <string>
#include <bitset>

 class FibLFSR{

 public:
        FibLFSR(const std::string& seed); //constructor to create LFSR with
                                     //the given initial seed and tap

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

        int generate(int k);         // simulate k steps and return
                                     // k-bit integer
        friend std::ostream& operator <<(std::ostream& lhs, const FibLFSR& rhs);
        //Overload the << stream insertion operator to display its
        //current register value in printable form


 private:
        std::bitset<16> value;

 };

```

FibLFSR.cpp

```cpp
#include "FibLFSR.h"
#include <stdlib.h>
#include <stdio.h>
using namespace std;

FibLFSR::FibLFSR(const std::string &seed) : value{seed} {}

int FibLFSR::step(){// simulate one step and return the
                                // new bit as 0 or 1

      int newBit = value[15] ^ value[13] ^ value[12] ^ value[10];
      //shift value
      value = value << 1;

      value[0] = newBit;

      return newBit;

  }

  int FibLFSR::generate(int k){// simulate k steps and return
                                  // k-bit integer
      int retVal = 0;
      int newVal = 0;
      for(int i = 0; i < k; i++){

          newVal = step();
          retVal = (retVal *2) + newVal; //double the return value and add the newvalue after calling step function

      }
      return retVal;

  }
   std::ostream& operator <<(std::ostream& out, const FibLFSR& rhs){

       out << rhs.value;
       return out;

  }

```

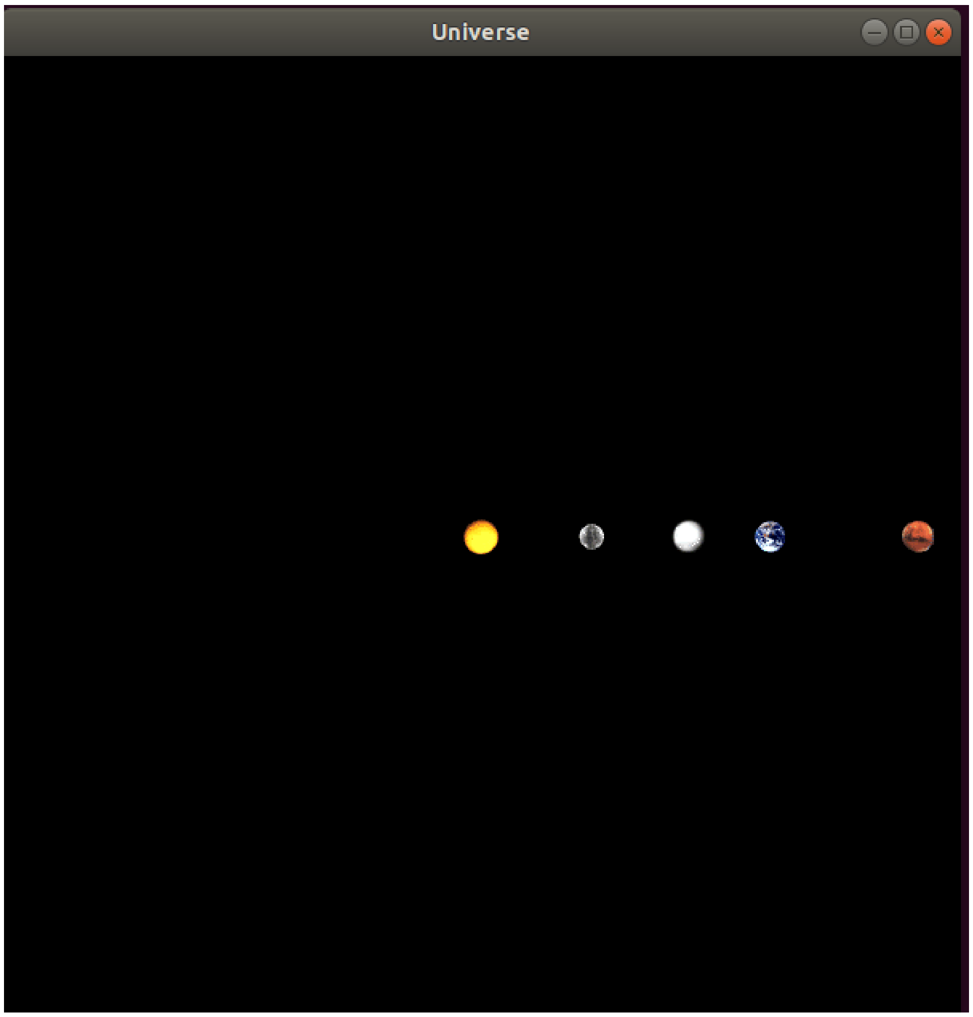
PS2a: N-Body Simulation

Assignment
This assignment implements a Universe class containing celestial bodies with the Sun, Mercury, Venus, and Mars which was read in through the command line. The implementation set each celestial body in their positions in which the class was used in the main to display each planet in a SFML window in their correct positions.

Key Concepts
This assignment used C++ and SFML features such as class implentations and SFML features to load in the images and display it in the window. The bodies were read in using input redirect < command line operator. The cin << operator was used to read in all of the data which was used to set and display each planet and sun in their correct positions. The coordinates of the bodies were done by using each body's radius and setting the new position with the new scaled coordinates so that SFML can display it in the window without each planet going out of bounds. The class also had to be sf::drawable in which the draw function was overloaded to draw the bodies.

What I Learned
I already knew the basics of SFML such as loading images, sprites, texteres, and such, but I had to learn how to implement the draw function by looking at the SFML website.

Screenshot:



Makefile:

```
CC = g++
CFLAGS = -std=c++17 -c -g -Og -Wall -Werror -pedantic
OBJ = NBody.o universe.o
DEPS = universe.cpp Nbody.cpp universe.hpp
LIBS = -lsfml-graphics -lsfml-window -lsfml-system
EXE = NBody

all: $(OBJ)
  $(CC) $(OBJ) -o $(EXE) $(LIBS)

 %.o: %.cpp $(DEPS)
         $(CC) $(CFLAGS) -o $@ $<

 clean:
         rm $(OBJ) $(EXE)

```

NBody.cpp

```cpp
// Copyright Andy Pen 2021
#include "universe.hpp"

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

int main(int argc, char *argv[]) {
    int newBody;
    float xPos;
    float yPos;
    float xVel;
    float yVel;
    float mass;
    float radius;
    std::string fileName;
    std::cin >> newBody;
    std::cin >> radius;

    std::vector<std::unique_ptr<CelestialBody>> nBody;
    for (int i = 0; i < newBody; ++i) {
        std::cin >> xPos >> yPos >> xVel >> yVel >> mass >> fileName;
        std::unique_ptr<CelestialBody> temp = std::make_unique<CelestialBody>(radius, sf::Vector2u(600, 600), xPos, yPos, xVel, yVel, mass, fileName);//NOLINT
        nBody.push_back(std::move(temp));
    }

    sf::RenderWindow window(sf::VideoMode(600, 600), "Universe");
    while (window.isOpen()) {
        for (sf::Event event; window.pollEvent(event);) {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear(sf::Color::Black);
        // print bodies
        for (int i = 0; i < newBody; ++i) {
            window.draw(nBody.at(i)->getSprite());
        }
        window.display();
    }
    return 0;
}

```

universe.hpp

```
// Copyright Andy Pen 2021
#include <iostream>
#include <string>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#ifndef _HOME_VISOTHPEN_COMP2040_PS2A_UNIVERSE_HPP_
#define _HOME_VISOTHPEN_COMP2040_PS2A_UNIVERSE_HPP_

 class CelestialBody : public sf::Drawable {
  public:
     CelestialBody();
     CelestialBody(float rad, sf::Vector2u window, float x, float y,
float xV, float yV, float m, std::string imgFile);//NOLINT

     // setter functions
     void setXpos(float x);
     void setYpos(float y);
     void setXvel(float xV);
     void setYvel(float yV);
     void setMass(float m);
     void setRadius(float r);
     void setWindow(sf::Vector2u windowScale);
     void setImg(std::string imgFile);
     // getter functions
     float getXpos();
     float getYpos();
     float getXvel();
     float getYvel();
     float getMass();
     float getRadius();
     sf::Vector2u getWindow();
     sf::Sprite getSprite();
     std::string getImg();

  private:
     float xPos;
     float yPos;
     float xVel;
     float yVel;
     float mass;
     float radius;
     sf::Sprite sprite;
     sf::Texture texture;
     std::string img;
     sf::Vector2u window;
     virtual void draw(sf::RenderTarget& target, sf::RenderStates
states) const;//NOLINT
 };

 #endif  // _HOME_VISOTHPEN_COMP2040_PS2A_UNIVERSE_HPP_

```

universe.cpp

```cpp
// Copyright Andy Pen 2021
#include "universe.hpp"
#include <iostream>
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>

CelestialBody::CelestialBody() {
    xPos = 0;
    yPos = 0;
    xVel = 0;
    yVel = 0;
    mass = 0;
    img = "";
}

CelestialBody::CelestialBody(float rad, sf::Vector2u window, float x, float y, float xV, float yV, float m, std::string imgFile) { //NOLINT
    xPos = x;
    yPos = y;
    xVel = xV;
    yVel = yV;
    mass = m;
    radius = rad;
    img = imgFile;

    sf::Image image;
    image.loadFromFile(img);
    texture.loadFromImage(image);
    sprite.setTexture(texture);

    //  set origin
    sprite.setOrigin(texture.getSize().x / 2, texture.getSize().y / 2);

    float xWindowRad = window.x / 2;
    float yWindowRad = window.y / 2;
    float xScale = (xWindowRad) / rad;
    float yScale = (yWindowRad) / rad;
    float newX = xPos * xScale + xWindowRad;
    float newY = yPos * yScale + yWindowRad;
    sprite.setPosition(newX, newY);
}

void CelestialBody::setXpos(float x) {
    xPos = x;
}

void CelestialBody::setYpos(float y) {
    yPos = y;
}

```

```cpp
void CelestialBody::setXvel(float xV) {
    xVel = xV;
}

void CelestialBody::setYvel(float yV) {
    yVel = yV;
}

void CelestialBody::setMass(float m) {
    mass = m;
}

void CelestialBody::setRadius(float r) {
    radius = r;
}

void CelestialBody::setWindow(sf::Vector2u windowScale) {
    window = windowScale;
}

void CelestialBody::setImg(std::string imgFile) {
    img = imgFile;
}

float CelestialBody::getXpos() {
    return xPos;
}

float CelestialBody::getYpos() {
    return yPos;
}

float CelestialBody::getXvel() {
    return xVel;
}

float CelestialBody::getYvel() {
    return yVel;
}

float CelestialBody::getMass() {
    return mass;
}

float CelestialBody::getRadius() {
    return radius;
}

sf::Vector2u CelestialBody::getWindow() {
    return window;
}

sf::Sprite CelestialBody::getSprite() {
    return sprite;
}

std::string CelestialBody::getImg() {
    return std::string();
}

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

```

PS2b: N-Body Simulation – Simulating the Universe

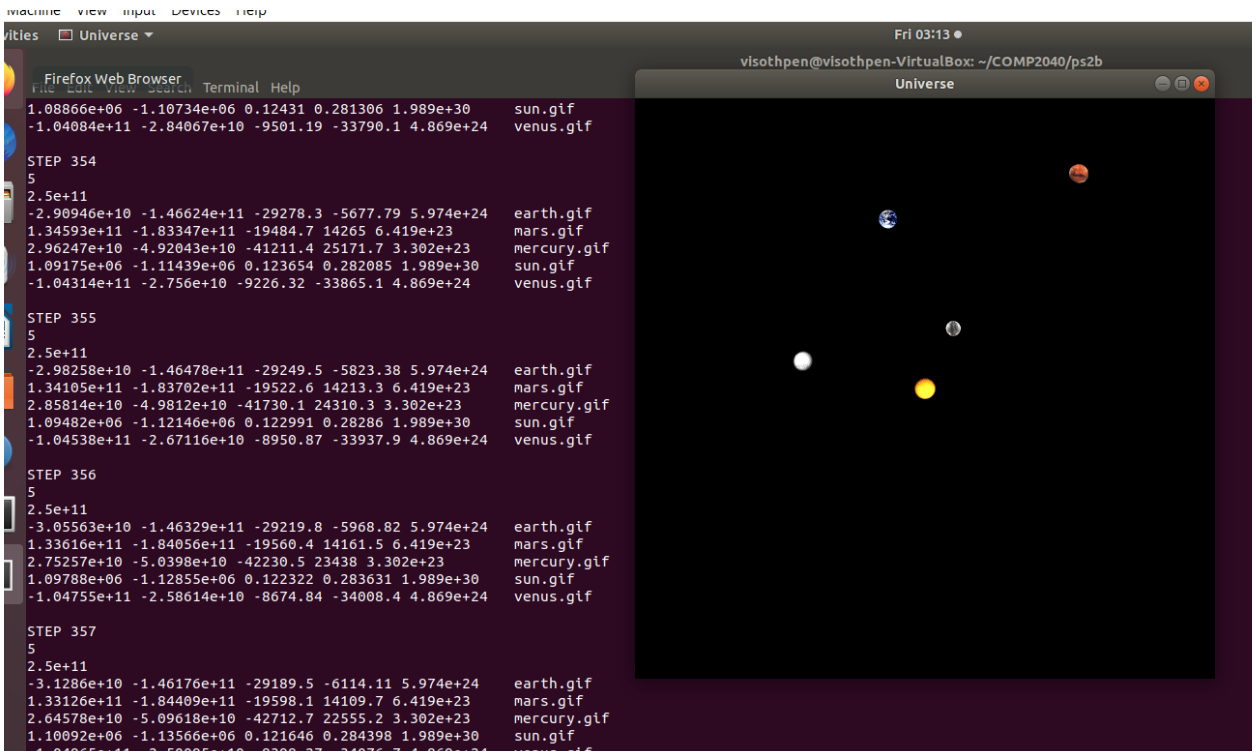
Assignment
This assignment is derived from the previous assignment, but I added two more functions to simulate the universe that would have the planets revolve in a counter-clockwise direction with the sun fixated in the center. I also had a function to print the positions of each body for each step that was made.

Key Concepts
This assignment used the previous assingment's class implenetations, but this time it was more math and physics that were involved. Newton's laws and formulas were used to simulate the planets moving. I used the following formulas given in the instructions:

```
F = (G * M1 * M2) / R^2
R = square root (R2)
R2 = (Δx)^2 + (Δy)^2
Δx = x2 - x1
Δy = y2 - y1
```

What I Learned
In this assignment, I learned mianly the physics behind the program in order to make the planets revolve counter-clockwise around the Sun that had to be fixated at the center of the window. I also had to make sure the planets didn't collide with each other or go out of bounds.

Screenshot:



```
1.08866e+06 -1.10734e+06 0.12431 0.281306 1.989e+30     sun.gif
-1.04084e+11 -2.84067e+10 -9501.19 -33790.1 4.869e+24   venus.gif

STEP 354
5
2.5e+11
-2.90946e+10 -1.46624e+11 -29278.3 -5677.79 5.974e+24   earth.gif
1.34593e+11 -1.83347e+11 -19484.7 14265 6.419e+23       mars.gif
2.96247e+10 -4.92043e+10 -41211.4 25171.7 3.302e+23     mercury.gif
1.09175e+06 -1.11439e+06 0.123654 0.282085 1.989e+30    sun.gif
-1.04314e+11 -2.756e+10 -9226.32 -33865.1 4.869e+24     venus.gif

STEP 355
5
2.5e+11
-2.98258e+10 -1.46478e+11 -29249.5 -5823.38 5.974e+24   earth.gif
1.34105e+11 -1.83702e+11 -19522.6 14213.3 6.419e+23     mars.gif
2.85814e+10 -4.9812e+10 -41730.1 24310.3 3.302e+23      mercury.gif
1.09482e+06 -1.12146e+06 0.122991 0.28286 1.989e+30     sun.gif
-1.04538e+11 -2.67116e+10 -8950.87 -33937.9 4.869e+24   venus.gif

STEP 356
5
2.5e+11
-3.05563e+10 -1.46329e+11 -29219.8 -5968.82 5.974e+24   earth.gif
1.33616e+11 -1.84056e+11 -19560.4 14161.5 6.419e+23     mars.gif
2.75257e+10 -5.0398e+10 -42230.5 23438 3.302e+23        mercury.gif
1.09788e+06 -1.12855e+06 0.122322 0.283631 1.989e+30    sun.gif
-1.04755e+11 -2.58614e+10 -8674.84 -34008.4 4.869e+24   venus.gif

STEP 357
5
2.5e+11
-3.1286e+10 -1.46176e+11 -29189.5 -6114.11 5.974e+24    earth.gif
1.33126e+11 -1.84409e+11 -19598.1 14109.7 6.419e+23     mars.gif
2.64578e+10 -5.09618e+10 -42712.7 22555.2 3.302e+23     mercury.gif
1.10092e+06 -1.13566e+06 0.121646 0.284398 1.989e+30    sun.gif
```

Makefile:

```
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
OBJ = NBody.o universe.o
DEPS = universe.cpp Nbody.cpp universe.hpp
LIBS = -lsfml-graphics -lsfml-window -lsfml-system
EXE = NBody

all: $(OBJ)
  $(CC) $(OBJ) -o $(EXE) $(LIBS)

 %.o: %.cpp $(DEPS)
         $(CC) $(CFLAGS) -o $@ $<

 clean:
         rm $(OBJ) $(EXE)

```

NBody.cpp

```
// Copyright Andy Pen 2021
#include "universe.hpp"
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>

int main(int argc, char *argv[]) {
    const double G = 6.67e-11;
    const double totalTime = std::stoi(argv[1]);
      const double stepTime = std::stoi(argv[2]);
      std::cout << G << totalTime << stepTime << std::endl;

      int newBody;
      float radius;
      float xPos;
      float yPos;
      float xVel;
      float yVel;
      float mass;
      int elapsedTime;
      double xForce;
      double yForce;
      double dX, dY;
      double distance;
      double netForce;
      double xAcc;
      double yAcc;
      double xVelocity;
      double yVelocity;
      std::string fileName;
      std::cin >> newBody;
      std::cin >> radius;

      std::vector<std::unique_ptr<CelestialBody>> nBody;
      for (int i = 0; i < newBody; ++i) {
          std::cin >> xPos >> yPos >> xVel >> yVel >> mass >> fileName;
          std::unique_ptr<CelestialBody> temp =
std::make_unique<CelestialBody>(radius, sf::Vector2u(600, 600), xPos,
yPos, xVel, yVel, mass, fileName);//NOLINT
          nBody.push_back(std::move(temp));
      }

      elapsedTime = 0;
      sf::RenderWindow window(sf::VideoMode(600, 600), "Universe");
      while (window.isOpen() && elapsedTime < totalTime) {
          for (sf::Event event; window.pollEvent(event);) {
              if (event.type == sf::Event::Closed)
                  window.close();
          }
          window.setFramerateLimit(60);
          window.clear(sf::Color::Black);
 // calculate distance, netforce, accel
```

```cpp
          for (int i = 0; i < newBody; i++) {
              xForce = 0;
              yForce = 0;
              for (int j = 0; j < newBody; j++) {
                  if (i != j) {
                      dX = nBody.at(j)->getXpos() - nBody.at(i)->getXpos();
                      dY = nBody.at(j)->getYpos() - nBody.at(i)->getYpos();
                      distance = sqrt((dX * dX) + (dY * dY));
                      netForce = (G * nBody.at(i)->getMass() * nBody.at(j)->getMass()) / (distance * distance); //NOLINT
                      xForce += netForce * (dX / distance);
                      yForce += netForce * (dY / distance);
                  }
              }
              xAcc = xForce / nBody.at(i)->getMass();
              yAcc = yForce / nBody.at(i)->getMass();

  // calculate vel
              xVelocity = nBody.at(i)->getXvel() + (stepTime * xAcc);
              yVelocity = nBody.at(i)->getYvel() - (stepTime * yAcc);
              nBody.at(i)->setXvel(xVelocity);
              nBody.at(i)->setYvel(yVelocity);
          }

  // print bodies
          for (int i = 0; i < newBody; ++i) {
              nBody.at(i)->step(stepTime);
              window.draw(nBody.at(i)->getSprite());
          }
          window.display();
          elapsedTime++;

          std::cout << "\nSTEP " << elapsedTime << std::endl;
          std::cout << newBody << std::endl;
          std::cout << radius << std::endl;
          for (int i = 0; i < newBody; ++i) {
              nBody.at(i)->printBody();
          }
      }
      // display times
      std::cout << "\nElapsed Time: " << elapsedTime << std::endl;
      std::cout << "Total Time: " << totalTime << std::endl;
      return 0;
  }

```

universe.hpp

```cpp
// Copyright Andy Pen 2021
#include <iostream>
#include <string>
#include <iomanip>
#include <fstream>
#include <vector>
#include <cmath>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
 #include <SFML/Graphics.hpp>
 #ifndef _HOME_VISOTHPEN_COMP2040_PS2B_UNIVERSE_HPP_
 #define _HOME_VISOTHPEN_COMP2040_PS2B_UNIVERSE_HPP_

 class CelestialBody : public sf::Drawable {
  public:
     CelestialBody();
     CelestialBody(float rad, sf::Vector2u window, float x, float y,
float xV, float yV, float m, std::string imgFile);//NOLINT

     // setter functions
     void setXpos(float x);
     void setYpos(float y);
     void setXvel(float xV);
     void setYvel(float yV);
     void setMass(float m);
     void setRadius(float r);
     void setWindow(sf::Vector2u windowScale);
     void setImg(std::string imgFile);
     // getter functions
     float getXpos();
     float getYpos();
     float getXvel();
     float getYvel();
     float getMass();
     float getRadius();
     sf::Vector2u getWindow();
     sf::Sprite getSprite();
     std::string getImg();
     void step(const double s);
     void printBody();

  private:
     float xPos;
     float yPos;
     float xVel;
     float yVel;
     float mass;
     float radius;
     sf::Sprite sprite;
     sf::Texture texture;
     std::string img;
     sf::Vector2u window;
     virtual void draw(sf::RenderTarget& target, sf::RenderStates
  states) const;//NOLINT
};

#endif  // _HOME_VISOTHPEN_COMP2040_PS2B_UNIVERSE_HPP_

```

universe.cpp

```cpp
// Copyright Andy Pen 2021
#include "universe.hpp"
#include <iostream>
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>

CelestialBody::CelestialBody() {
    xPos = 0;
    yPos = 0;
    xVel = 0;
    yVel = 0;
    mass = 0;
    img = "";
}

CelestialBody::CelestialBody(float rad, sf::Vector2u window, float x, float y, float xV, float yV, float m, std::string imgFile) { //NOLINT
    xPos = x;
    yPos = y;
    xVel = xV;
    yVel = yV;
    mass = m;
    radius = rad;
    img = imgFile;

    sf::Image image;
    image.loadFromFile(img);
    texture.loadFromImage(image);
    sprite.setTexture(texture);

    //  set origin
    sprite.setOrigin(texture.getSize().x / 2, texture.getSize().y / 2);

    float xWindowRad = window.x / 2;
    float yWindowRad = window.y / 2;
    float xScale = (xWindowRad) / rad;
    float yScale = (yWindowRad) / rad;
    float newX = xPos * xScale + xWindowRad;
    float newY = yPos * yScale + yWindowRad;
    sprite.setPosition(newX, newY);
}

void CelestialBody::setXpos(float x) {
    xPos = x;
}

void CelestialBody::setYpos(float y) {
    yPos = y;
}

```

```cpp
void CelestialBody::setXvel(float xV) {
    xVel = xV;
}

void CelestialBody::setYvel(float yV) {
    yVel = yV;
}

void CelestialBody::setMass(float m) {
    mass = m;
}

void CelestialBody::setRadius(float r) {
    radius = r;
}

void CelestialBody::setWindow(sf::Vector2u windowScale) {
    window = windowScale;
}

void CelestialBody::setImg(std::string imgFile) {
    img = imgFile;
}

float CelestialBody::getXpos() {
    return xPos;
}

float CelestialBody::getYpos() {
    return yPos;
}

float CelestialBody::getXvel() {
    return xVel;
}

float CelestialBody::getYvel() {
    return yVel;
}

float CelestialBody::getMass() {
    return mass;
}

float CelestialBody::getRadius() {
    return radius;
}

sf::Vector2u CelestialBody::getWindow() {
    return window;
}

sf::Sprite CelestialBody::getSprite() {
    return sprite;
}

std::string CelestialBody::getImg() {
    return std::string();
}

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

void CelestialBody::step(const double s) {
    xPos += s * xVel;
    yPos -= s * yVel;
    double xWinRadius = 600 / 2;
    double yWinRadius = 600 / 2;
    double xScale = (xWinRadius) / radius;
    double yScale = (yWinRadius) / radius;

    double newX = (xPos * xScale) + xWinRadius;
    double newY = (yPos * yScale) + yWinRadius;

    sprite.setPosition(newX, newY);
}

void CelestialBody::printBody() {
    std::cout << xPos << " " << yPos << " " << xVel << " " << yVel << " " << mass << " \t" << img << std::endl;//NOLINT
}

```

PS3: DNA Sequence Alignment

Assignment

This assignment uses dynamic programming to compute the optimal sequence alignment of two DNA strings. With the two DNA strings, I implemented a edit distance class that compared the two strings and checked for penalties, matches, and mismatches that would determine the cost.

Key Concepts

This assignment used dynamic programming in which the Needleman-Wunsch method was used to make a NxM matrix. I had to calculate the edit distances of the DNA strings until it traced back to the position [0][0] of the matrix.The alignment was recovered by looking at three cases which were:

Case 1. The optimal alignment matches x[i] up with y[j]. In this case, we must have opt[i][j] = opt[i+1][j+1] if x[i] equals y[j], or opt[i][j] = opt[i+1][j+1] + 1 otherwise.

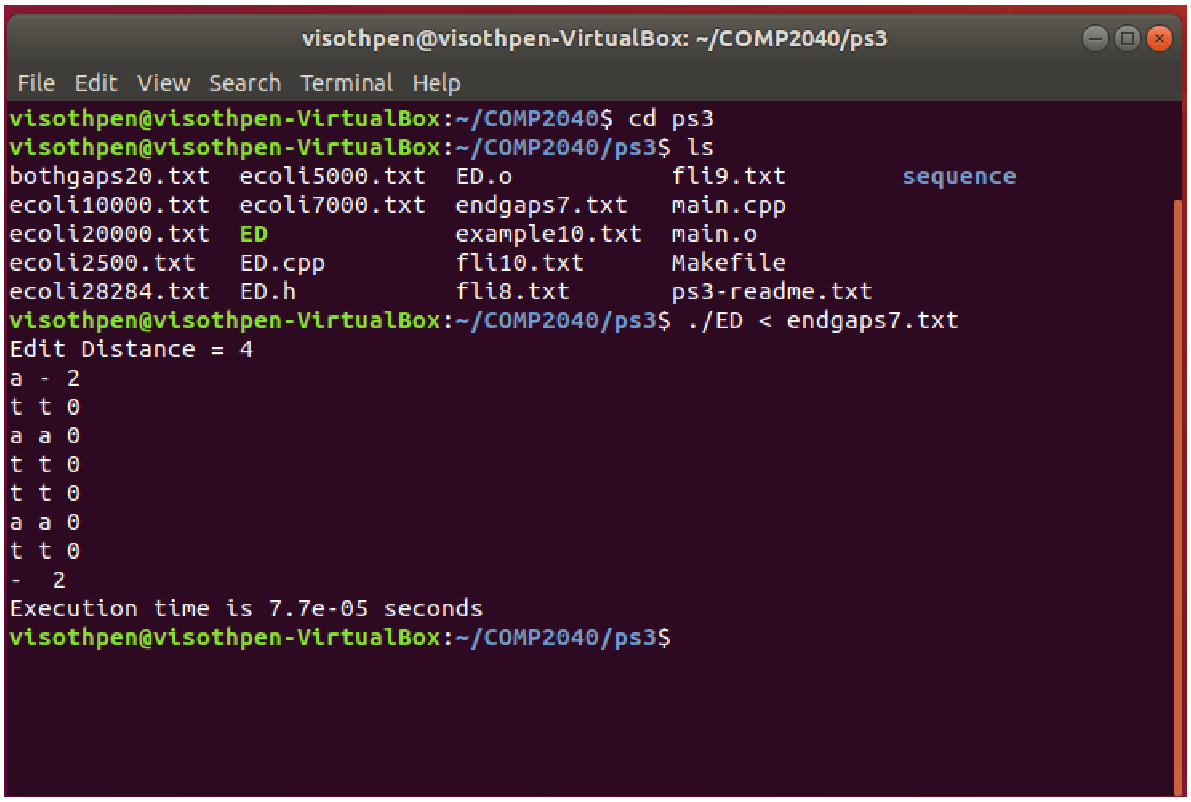
Case 2. The optimal alignment matches x[i] up with a gap. In this case, we must have opt[i][j] = opt[i+1][j] + 2.

Case 3. The optimal alignment matches y[j] up with a gap. In this case, we must have opt[i][j] = opt[i][j+1] + 2.

What I Learned

I learned how to use dynamic programming to create a matrix that was used to find the cost between two DNA string alignments. I haven’t used matrixes much, but this assignment made me learn more about the [N][M] matrix used and traversing through it.

Screenshot:



visothpen@visothpen-VirtualBox: ~/COMP2040/ps3
File Edit View Search Terminal Help

```
visothpen@visothpen-VirtualBox:~/COMP2040$ cd ps3
visothpen@visothpen-VirtualBox:~/COMP2040/ps3$ ls
bothgaps20.txt  ecoli5000.txt  ED.o           fli9.txt        sequence
ecoli10000.txt  ecoli7000.txt  endgaps7.txt   main.cpp
ecoli20000.txt  ED             example10.txt  main.o
ecoli2500.txt   ED.cpp         fli10.txt      Makefile
ecoli28284.txt  ED.h           fli8.txt       ps3-readme.txt
visothpen@visothpen-VirtualBox:~/COMP2040/ps3$ ./ED < endgaps7.txt
Edit Distance = 4
a - 2
t t 0
a a 0
t t 0
t t 0
a a 0
t t 0
-   2
Execution time is 7.7e-05 seconds
visothpen@visothpen-VirtualBox:~/COMP2040/ps3$
```

Makefile:

```makefile
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
OBJ = main.o ED.o
DEPS = ED.cpp ED.h main.cpp
LIBS = -lsfml-graphics -lsfml-system
EXE = ED

all: $(OBJ)
  $(CC) $(OBJ) -o $(EXE) $(LIBS)

%.o: %.cpp $(DEPS)
        $(CC) $(CFLAGS) -o $@ $<

clean:
        rm $(OBJ) $(EXE)

```

main.cpp

```cpp
#include "ED.h"
#include <SFML/System.hpp>

int main(int argc, char* argv[]){
    //read in 2 stirngs
    std::string str1;
    std::string str2;
    std::cin >> str1;
    std::cin >> str2;

      sf::Clock clock;
      sf::Time t;

      ED ed(str1, str2);

      //display edit distance
      int editDistance;
      editDistance = ed.OptDistance();

      std::cout << "Edit Distance = " << editDistance << std::endl;

      std::cout << ed.Alignment();
  //end of main, after computing solution, capture running time
      t = clock.getElapsedTime();

  //display running time
      std::cout << "Execution time is " << t.asSeconds() << " seconds
\n";

      return 0;
  }

```

ED.h

```
#include <iostream>
#include <string>
#include <stdlib.h>
#include <vector>
#include <sstream>

class ED{
public:
    ED(std::string x, std::string y);
      int penalty(char a, char b);
      int min(int a, int b, int c);
      int OptDistance();
      std::string Alignment();

 private:
      std::string X;
      std::string Y;
      std::vector <std::vector<int>> matrix;

 };

```

ED.cpp

```
#include "ED.h"
#include <algorithm>

ED::ED(std::string x, std::string y){
   //accept 2 strings to be compared
    //allocate any data structures
    X = x;
    Y = y;
    //make sure there is enough space
      matrix = std::vector <std::vector<int>> ((int)X.length() +1);
      for(int i =0; i <= (int)X.length(); i++){
          matrix[i] = std::vector<int>((int)Y.length() + 1);
      }

  }


  int ED::penalty(char a, char b){
      //return the penalty for aligning chars a and b (0 or 1)
      if(a == b){
          return 0;
      }
      else{
          return 1;
      }

  }

  int ED::min(int a, int b, int c){
      //return the minimum of three arguments
      return std::min({a,b,c}); //included in <algorithm>

  }

  int ED::OptDistance(){
      //populates matrix based on two strings
      //return optimal distance
      int M; //rows
      int N; //columns
      M = X.length();
      N = Y.length();

      for(int i = 0; i <= M; i++){
          for(int j = 0; j <= N; j++){
              matrix[M][j] = 2 * (N-j);
          }
          matrix[i][N] = 2 * (M-i);
      }

      for(int i = M - 1; i >= 0; i--){
          for(int j = N - 1; j >= 0; j--){
```

```cpp
            matrix[i][j] = min(matrix[i+1][j+1] + penalty(X[i], Y[j]), matrix[i+1][j] + 2, matrix[i][j+1] + 2);
        }
    }


    //return optimal distance at matrix[0][0]
    return matrix[0][0];
}

std::string ED::Alignment(){
    //traces matrix

    //use string stream to read in string as stream
    std::stringstream output;

    int newX;
    int newY;
    newX = X.length();
    newY = Y.length();

    int i = 0;
    int j = 0;

//create table
//end gap = 2
// match = 0
//mismatch = 1
// retrace from from opt[0][0] to opt[M][N]
    while(i < newX && j < newY){
        if(matrix[i][j] - 2 == matrix[i][j+1]){
            output << "- " << Y[j] << " 2" << std::endl;
            j++;
        }
        else if(matrix[i][j] - 2 == matrix[i+1][j]){
            output << X[i] << " - 2" << std::endl;
            i++;
        }
        else if(matrix[i][j] - 1 == matrix[i+1][j+1]){
            output << X[i] << ' ' << Y[j] << " 1" << std::endl;
            i++;
            j++;
        }
        else{
            output << X[i] << ' ' << Y[j] << " 0" << std::endl;
            i++;
            j++;
        }
    }

    if(i < newX){
        output << X[i+1] << " - 2" << std::endl;
    }
    else if(j < newY){
        output << "- " << Y[j+1] << " 2" << std::endl;
    }

    //return string that can be printed to display actual alignment
    return output.str();
}

```

PS4a: Synthesizing a Plucked String Sound – Circular Buffer Implementation

Assignment

This assignment was the first part to synthesizing a plucked string sound. I implemented a CircularBuffer class which created an empty buffer array that add an item to the end of the array while returning an item to the front. I also had to include throw exceptions to check for invalid arguments and runtime errors. Boost test framework was also used to test the implementation,

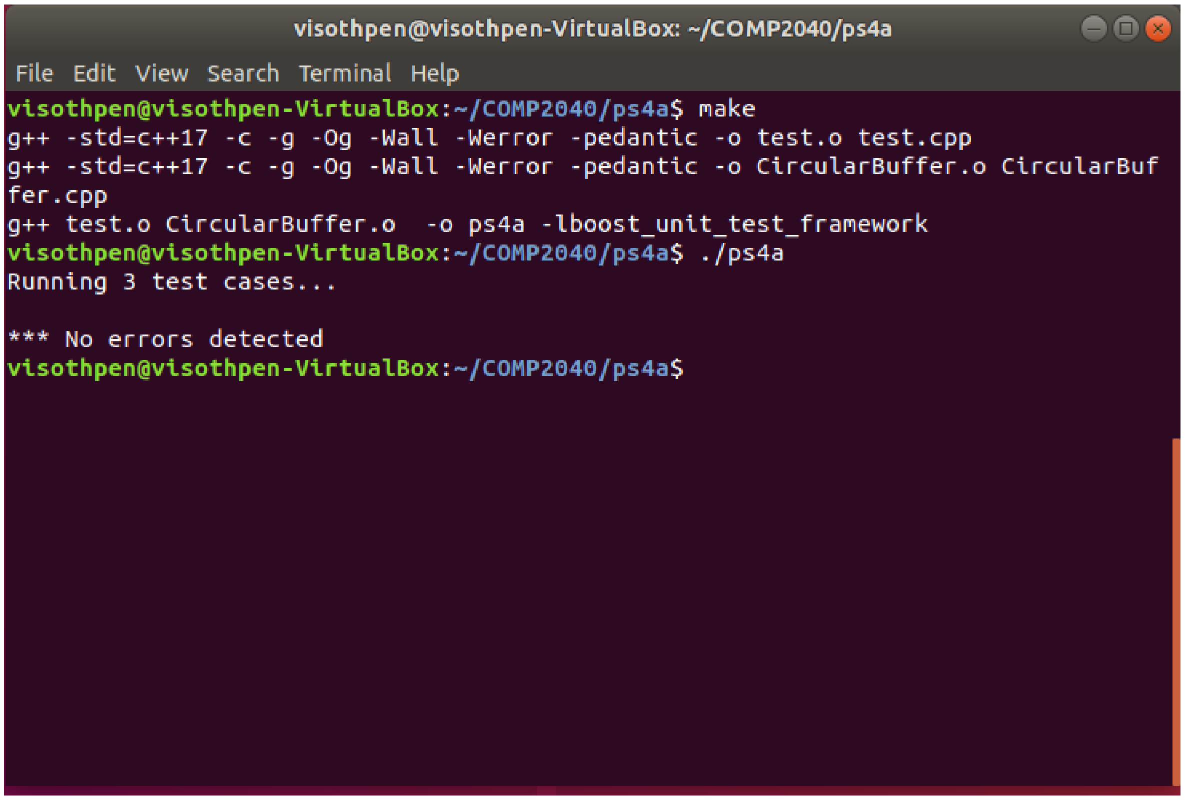
Key Concepts

This assignment used a circular buffer array that created an empty buffer array based on the capacity and checked whether or not if it was full. It would add an item to the end and return an time to the front instead of deleting it. Throw exceptions were used in case there were any invalid arguments such as the capacity or size being too little or negative or runtime error if the array can't enque properly. Boost test cases were used to test the implementation. Also, cpplint style check was used to check the style of the code and make it more neater.

What I Learned

In this assignment, I learned how circular buffer arrays worked and how to implement it. The implementation was similar to how I would have to implement queues back in COMPII. I also learned how to properly use throw exceptions to check for any runtime errors or invalid arguments. Lastly, I learned how to use the cpplint feature to check the style of the code. It made my code more neat and follow the style guidelines to pass the cpplint test. I already knew how to use the Boost tests from assignment PS1a so I basically went based off of that for the unit testing.

Screenshot:


visothpen@visothpen-VirtualBox: ~/COMP2040/ps4a
File Edit View Search Terminal Help

```
visothpen@visothpen-VirtualBox:~/COMP2040/ps4a$ make
g++ -std=c++17 -c -g -Og -Wall -Werror -pedantic -o test.o test.cpp
g++ -std=c++17 -c -g -Og -Wall -Werror -pedantic -o CircularBuffer.o CircularBuf
fer.cpp
g++ test.o CircularBuffer.o  -o ps4a -lboost_unit_test_framework
visothpen@visothpen-VirtualBox:~/COMP2040/ps4a$ ./ps4a
Running 3 test cases...

*** No errors detected
visothpen@visothpen-VirtualBox:~/COMP2040/ps4a$
```

Makefile:

```
CC = g++
CFLAGS = -std=c++17 -c -g -Og -Wall -Werror -pedantic
OBJ = test.o CircularBuffer.o
DEPS = CircularBuffer.h CircularBuffer.cpp test.cpp
LIBS = -lboost_unit_test_framework
EXE = ps4a

all: $(OBJ)
  $(CC) $(OBJ) -o $(EXE) $(LIBS)

 %.o: %.cpp $(DEPS)
        $(CC) $(CFLAGS) -o $@ $<

 clean:
        rm $(OBJ) $(EXE)

```

test.cpp

```
// (C) Copyright Andy Pen, 2021.
#include "CircularBuffer.h"
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(firstTestCase) {
CircularBuffer CB(7);
BOOST_REQUIRE(CB.isEmpty());

CB.enqueue(8);
CB.enqueue(3);
CB.enqueue(1);
BOOST_REQUIRE_EQUAL(CB.dequeue(), 8);
BOOST_REQUIRE_EQUAL(CB.dequeue(), 3);
BOOST_REQUIRE_EQUAL(CB.dequeue(), 1);
}

BOOST_AUTO_TEST_CASE(secondTestCase) {
CircularBuffer CB(5);
BOOST_REQUIRE_THROW(CB.dequeue(), std::runtime_error);
}

BOOST_AUTO_TEST_CASE(thirdTestCase) {
CircularBuffer CB(10);

BOOST_REQUIRE_NO_THROW(CB.enqueue(5));
}

```

CircularBuffer.h

```cpp
// (C) Copyright Andy Pen, 2021.
#include <iostream>
#include <stdint.h>  // defines standard 16-bit integer type int16_t.
#include <memory>
#include <stdexcept>

class CircularBuffer {
 public:
    CircularBuffer(int capacity);
      int size();
      bool isEmpty();
      bool isFull();
      void enqueue(int16_t x);
      int16_t dequeue();
      int16_t peek();

  private:
      int front;
      int rear;
      int buffCap;
      int buffSize;
      std::unique_ptr<std::int16_t[]> arr;
 };

```

CircularBuffer.cpp

```cpp
// (C) Copyright Andy Pen, 2021.

#include "CircularBuffer.h"

CircularBuffer::CircularBuffer(int capacity) {
// create an empty ring buffer, with given max capacity
    front = 0;
    rear = 0;
    buffSize = 0;
    buffCap = capacity;
    if (buffCap <= 0) {
        throw std::invalid_argument("capacity must be greater than zero.");
    }
    // create buffer ring
    arr = std::make_unique<std::int16_t[]> (buffCap);
}

int CircularBuffer::size() {
// return number of items currently in the buffer
    return buffSize;
}

bool CircularBuffer::isEmpty() {
// is the buffer empty (size equals zero)?
    return (buffSize == 0);
}

bool CircularBuffer::isFull() {
// is the buffer full (size equals capacity)?
    return (buffSize == buffCap);
}


void CircularBuffer::enqueue(int16_t x) {
// add item x to the end
    if (isFull()) {
        throw std::runtime_error("can't enqueue to a full ring");
    } else {
        arr[rear] = x;
        rear = (rear +1) % buffCap;
        // increment size of buffer
        buffSize++;
    }
}

int16_t CircularBuffer::dequeue() {
// delete and return item from the front
    if (isEmpty()) {
        throw std::runtime_error("Buffer is empty.");
    }
    auto temp = arr[front];
    front = (front + 1) % buffCap;
    // decrement size of buffer
    buffSize--;
    return temp;
}

int16_t CircularBuffer::peek() {
// return (but do not delete) item from the front

    if (isEmpty()) {
        throw std::runtime_error("Buffer is empty.");
    }
    return arr[front];
}

```

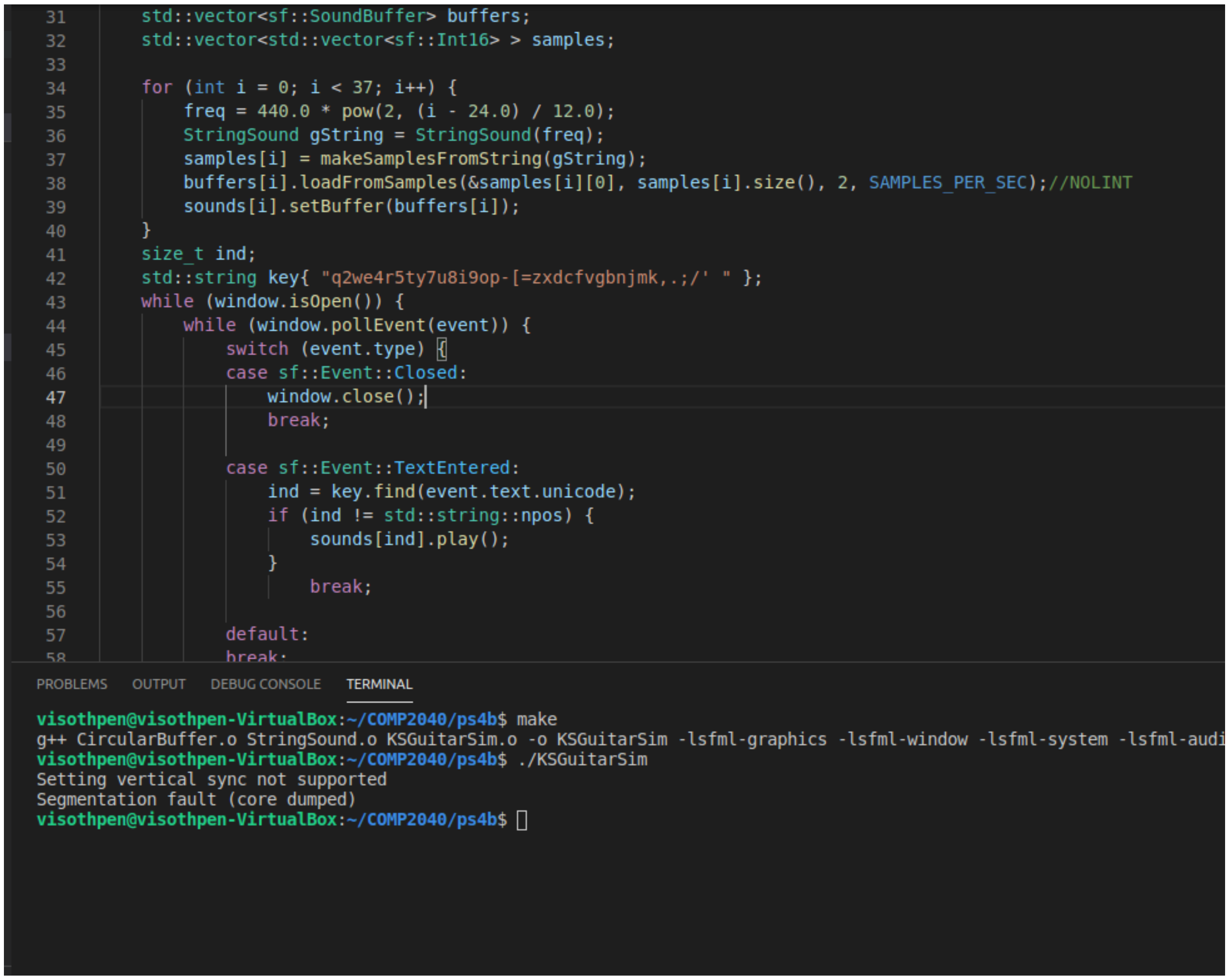
PS4b: Synthesizing a Plucked String Sound – SFML Audio Output

Assignment
This assignment was derived from the previous assignment that implemented the circularbuffer arrays. A StringSound class was implemented to create a guitar string sound using the 37 keys to represent the keyboard. However, I wasn't able to generate the string sounds because my program had a segmentation fault.

Key Concepts
This assignment uses vectos of sound samples in which the string sound is created fro ma vector of sf::Int16s. SFML is used to generate an electronic keyboard that would play a sound whenever a key is pressed on the keyboard. The new operator and ceiling function were also used in the StringSound class implementation.

What I Learned
This assignment used the features of SFML, but this time sound was introduced. Even though, I couldn't get the sound to work, I learned how to set the keys to generate the sound whenever the key was pressed.

Screenshot:



```
    std::vector<sf::SoundBuffer> buffers;
    std::vector<std::vector<sf::Int16> > samples;

    for (int i = 0; i < 37; i++) {
        freq = 440.0 * pow(2, (i - 24.0) / 12.0);
        StringSound gString = StringSound(freq);
        samples[i] = makeSamplesFromString(gString);
        buffers[i].loadFromSamples(&samples[i][0], samples[i].size(), 2, SAMPLES_PER_SEC);//NOLINT
        sounds[i].setBuffer(buffers[i]);
    }
    size_t ind;
    std::string key{ "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' " };
    while (window.isOpen()) {
        while (window.pollEvent(event)) {
            switch (event.type) {
            case sf::Event::Closed:
                window.close();
                break;

            case sf::Event::TextEntered:
                ind = key.find(event.text.unicode);
                if (ind != std::string::npos) {
                    sounds[ind].play();
                }
                    break;

            default:
            break;
```

```
visothpen@visothpen-VirtualBox:~/COMP2040/ps4b$ make
g++ CircularBuffer.o StringSound.o KSGuitarSim.o -o KSGuitarSim -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audi
visothpen@visothpen-VirtualBox:~/COMP2040/ps4b$ ./KSGuitarSim
Setting vertical sync not supported
Segmentation fault (core dumped)
visothpen@visothpen-VirtualBox:~/COMP2040/ps4b$
```

Makefile:

```makefile
CC = g++
CFLAGS = -std=c++17 -c -g -Og -Wall -Werror -pedantic
OBJ = CircularBuffer.o StringSound.o KSGuitarSim.o
DEPS = CircularBuffer.h CircularBuffer.cpp StringSound.cpp StringSound.h KSGuitarSim.cpp
LIBS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
EXE = KSGuitarSim

all: $(OBJ)
  $(CC) $(OBJ) -o $(EXE) $(LIBS)

%.o: %.cpp $(DEPS)
        $(CC) $(CFLAGS) -o $@ $<

clean:
        rm $(OBJ) $(EXE)

```

KSGuitarSim.cpp

```cpp
// (C) Copyright Andy Pen, 2021.
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>

#include "CircularBuffer.h"
#include "StringSound.h"

#define CONCERT_A 220.0
#define SAMPLES_PER_SEC 44100

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(300, 200), "SFML Guitar Hero Lite");
    sf::Event event;
    double freq;
    std::vector<sf::Sound> sounds;
    std::vector<sf::SoundBuffer> buffers;
    std::vector<std::vector<sf::Int16> > samples;

    for (int i = 0; i < 37; i++) {
        freq = 440.0 * pow(2, (i - 24.0) / 12.0);
        StringSound gString = StringSound(freq);
        samples[i] = makeSamplesFromString(gString);
        buffers[i].loadFromSamples(&samples[i][0], samples[i].size(), 2, SAMPLES_PER_SEC);//NOLINT
        sounds[i].setBuffer(buffers[i]);
    }
    size_t ind;
    std::string key{ "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' " };
    while (window.isOpen()) {
        while (window.pollEvent(event)) {
            switch (event.type) {
            case sf::Event::Closed:
                window.close();
                break;

            case sf::Event::TextEntered:
                ind = key.find(event.text.unicode);
                if (ind != std::string::npos) {
                    sounds[ind].play();
                }
                    break;

            default:
            break;
            }
            window.clear();
            window.display();
        }
    }
    return 0;
}

```

StringSound.h

```
#include "CircularBuffer.h"
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>

#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include <vector>

#include <math.h>
#include <limits.h>

class StringSound {
 public:
    explicit StringSound (double frequency);
    explicit StringSound (std::vector<sf::Int16> init);
    StringSound (const StringSound &obj) {};
    ~StringSound();
    void pluck();
    void tic();
    sf::Int16 sample();
    int time();
 private:
    CircularBuffer * _cb;
    int _time;
};

```

StringSound.cpp

```cpp
// (C) Copyright Andy Pen, 2021.
#include "StringSound.h"
#include "CircularBuffer.h"
constexpr unsigned sample_rate = 44100;
StringSound::StringSound(double frequency) {
// create a guitar string sound of the
// given frequency using a sampling rate of 44,100
    _time = 0;
    int length;
     if (frequency <= 0) {
         throw std::invalid_argument("Capacity must be greater than zero.");
     } else {
         length = ceil(sample_rate / frequency);
         _cb = new CircularBuffer(length);
         while (!_cb->isFull()) {
             _cb->enqueue(0);
         }
     }
 }

 StringSound::StringSound(std::vector<sf::Int16> init) {
 // create a guitar string with size and initial values given by vector
     _time = 0;
     int count;
     if (init.size() <= 0) {
         throw std::invalid_argument("Capacity must be greater than zero.");
     } else {
         _cb = new CircularBuffer(init.size());
         count = 0;
         while (!_cb->isFull()) {
             _cb->enqueue(init[count]);
             count++;
         }
     }
 }

 StringSound::~StringSound() {}

 void StringSound::pluck() {
 // pluck the guitar string by replacing buffer w/ random values
 // representing white noise
     if (_cb->isFull()) {
         for (int i = 0; i < _cb->size(); i++) {
             _cb->dequeue();
         }
     }
     while (_cb->isFull()) {
         _cb->enqueue((sf::Int16)(rand() * 0xffff)); //NOLINT
     }
 }
```

```

void StringSound::tic() {
// advance the simulation one time step
    sf::Int16 tic;
    tic = 0.5 * 0.996 * (_cb->dequeue() + _cb->peek());
    _cb->enqueue(tic);
    _time++;
}

sf::Int16 StringSound::sample() {
// return the current sample
    return _cb->peek();
}

int StringSound::time() {
// return number of times tic was called so far
    return _time;
}

```

CircularBuffer.h

```
// (C) Copyright Andy Pen, 2021.
#include <iostream>
#include <stdint.h>  // defines standard 16-bit integer type int16_t.
#include <memory>
#include <stdexcept>

class CircularBuffer {
 public:
    CircularBuffer(int capacity);
    int size();
    bool isEmpty();
    bool isFull();
    void enqueue(int16_t x);
    int16_t dequeue();
    int16_t peek();

 private:
    int front;
    int rear;
    int buffCap;
    int buffSize;
    std::unique_ptr<std::int16_t[]> arr;
};

```

CircularBuffer.cpp

```cpp
// (C) Copyright Andy Pen, 2021.

#include "CircularBuffer.h"

CircularBuffer::CircularBuffer(int capacity) {
// create an empty ring buffer, with given max capacity
    front = 0;
    rear = 0;
    buffSize = 0;
    buffCap = capacity;
    if (buffCap <= 0) {
        throw std::invalid_argument("capacity must be greater than zero.");
    }
    // create buffer ring
    arr = std::make_unique<std::int16_t[]> (buffCap);
}

int CircularBuffer::size() {
// return number of items currently in the buffer
    return buffSize;
}

bool CircularBuffer::isEmpty() {
// is the buffer empty (size equals zero)?
    return (buffSize == 0);
}

bool CircularBuffer::isFull() {
// is the buffer full (size equals capacity)?
    return (buffSize == buffCap);
}


void CircularBuffer::enqueue(int16_t x) {
// add item x to the end
    if (isFull()) {
        throw std::runtime_error("can't enqueue to a full ring");
    } else {
        arr[rear] = x;
        rear = (rear +1) % buffCap;
        // increment size of buffer
        buffSize++;
    }
}

int16_t CircularBuffer::dequeue() {
// delete and return item from the front
    if (isEmpty()) {
        throw std::runtime_error("Buffer is empty.");
    }
    auto temp = arr[front];
    front = (front + 1) % buffCap;
    // decrement size of buffer
    buffSize--;
    return temp;
}

int16_t CircularBuffer::peek() {
// return (but do not delete) item from the front

    if (isEmpty()) {
        throw std::runtime_error("Buffer is empty.");
    }
    return arr[front];
}

```

PS5: Recursive Graphics (Triangle Fractal)

Assignment

This assignment writes a program to plot a triangle fractal. A Triangle class was implemented to be sf::drawable and was used to draw the traingles and set the color. The main function had a recursive function that recursively called the TFractal function to draw the triangles. The program read in two command line arguments that was used for the length of the triangles and the depth of recursion.

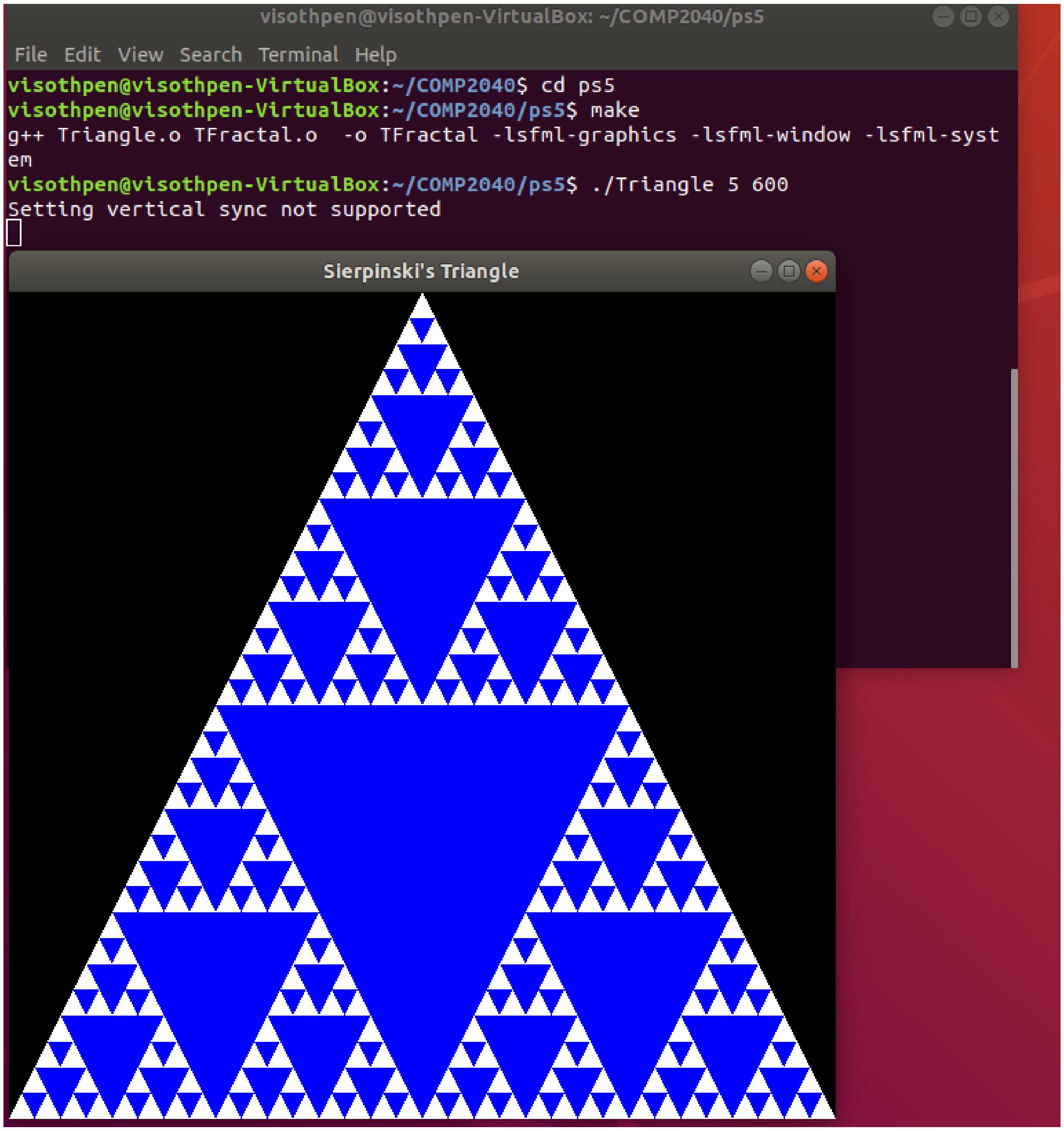
Key Concepts

This program used the concept of Sierpinksi's triangles to recursively draw triangles based on the input. SFML was used to display the triangle fractal and to change the color of the triangles.

What I Learned

In this assignment, sf::drawable was used again in which I already knew how to implement based on assignment ps2a. This assignment involved mostly math to correctly set the position of each points of the triangles. I had to touch up on recursion in which I used in the TFractal function.

Screenshot:



visothpen@visothpen-VirtualBox: ~/COMP2040/ps5
File Edit View Search Terminal Help

```
visothpen@visothpen-VirtualBox:~/COMP2040$ cd ps5
visothpen@visothpen-VirtualBox:~/COMP2040/ps5$ make
g++ Triangle.o TFractal.o  -o TFractal -lsfml-graphics -lsfml-window -lsfml-syst
em
visothpen@visothpen-VirtualBox:~/COMP2040/ps5$ ./Triangle 5 600
Setting vertical sync not supported
```

Makefile:

```
CC = g++
CFLAGS = -std=c++17 -c -g -Og -Wall -Werror -pedantic
OBJ = Triangle.o TFractal.o
DEPS = Triangle.cpp Triangle.h TFractal.cpp
LIBS = -lsfml-graphics -lsfml-window -lsfml-system
EXE = TFractal

all: $(OBJ)
	$(CC) $(OBJ) -o $(EXE) $(LIBS)

%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -o $@ $<

clean:
	rm $(OBJ) $(EXE)

```

TFractal.cpp

```cpp
// Copyright Andy Pen
#include "Triangle.h"
#include <SFML/System.hpp>
#include <SFML/Window.hpp>

// draw base triangle
void drawBase(int depth, int height, sf::RenderWindow &window); //NOLINT
// recursive function
void TFractal(int n, int depth, sf::Vector2f pt1, sf::Vector2f pt2, sf::Vector2f pt3, sf::RenderWindow &window); //NOLINT

int main(int argc, char *argv[]) {
  int N = std::stoi(argv[1]);
  double L = std::stod(argv[2]);

  sf::RenderWindow window(sf::VideoMode(L, L), "Sierpinski's Triangle");
  while (window.isOpen()) {
    sf::Event event;
    while (window.pollEvent(event)) {
      if (event.type == sf::Event::Closed) {
        window.close();
      }
    }

    drawBase(N, L, window);
    window.display();
  }

  return 0;
}

void drawBase(int n, int l, sf::RenderWindow &window) { //NOLINT
  sf::Vector2f pt1(l / 2, 0);
  sf::Vector2f pt2(0, l);
  sf::Vector2f pt3(l, l);

  Triangle triangle(pt1, pt2, pt3);
  window.draw(triangle);

  TFractal(1, n,
    sf::Vector2f((pt1.x + pt2.x) / 2, (pt1.y + pt2.y) / 2),
    sf::Vector2f((pt1.x + pt3.x) / 2, (pt1.y + pt3.y) / 2),
    sf::Vector2f((pt2.x + pt3.x) / 2, (pt2.y + pt3.y) / 2),
    window);
}

void TFractal(int size, int depth, sf::Vector2f pt1, sf::Vector2f pt2, sf::Vector2f pt3, sf::RenderWindow &window) { //NOLINT
  Triangle triangle(pt1, pt2, pt3);
  triangle.setColorBlue();
```

```
    window.draw(triangle);

      if (size < depth) {
          TFractal(size + 1, depth,
          sf::Vector2f((pt1.x + pt2.x) / 2 + (pt2.x - pt3.x) / 2,
          (pt1.y + pt2.y) / 2 + (pt2.y - pt3.y) / 2),
          sf::Vector2f((pt1.x + pt2.x) / 2 + (pt1.x - pt3.x) / 2,
          (pt1.y + pt2.y) / 2 + (pt1.y - pt3.y) / 2),
          sf::Vector2f((pt1.x + pt2.x) / 2, (pt1.y + pt2.y) / 2),
          window);

          TFractal(size + 1, depth,
          sf::Vector2f((pt3.x + pt2.x) / 2 + (pt2.x - pt1.x) / 2,
          (pt3.y + pt2.y) / 2 + (pt2.y - pt1.y) / 2),
          sf::Vector2f((pt3.x + pt2.x) / 2 + (pt3.x - pt1.x) / 2,
          (pt3.y + pt2.y) / 2 + (pt3.y - pt1.y) / 2),
          sf::Vector2f((pt3.x + pt2.x) / 2, (pt3.y + pt2.y) / 2),
          window);

          TFractal(size + 1, depth,
          sf::Vector2f((pt1.x + pt3.x) / 2 + (pt3.x - pt2.x) / 2,
          (pt1.y + pt3.y) / 2 + (pt3.y - pt2.y) / 2),
          sf::Vector2f((pt1.x + pt3.x) / 2 + (pt1.x - pt2.x) / 2,
          (pt1.y + pt3.y) / 2 + (pt1.y - pt2.y) / 2),
          sf::Vector2f((pt1.x + pt3.x) / 2, (pt1.y + pt3.y) / 2),
          window);
      }
  }

```

Triangle.h

```cpp
// Copyright Andy Pen
#include <iostream>
#include <SFML/Graphics.hpp>
#include <cstdlib>//NOLINT
#include <string>//NOLINT
#ifndef _HOME_VISOTHPEN_COMP2040_PS5_TRIANGLE_H_
#define _HOME_VISOTHPEN_COMP2040_PS5_TRIANGLE_H_

class Triangle : public sf::Drawable {
  public:
   Triangle(const sf::Vector2f& pt1, const sf::Vector2f& pt2, const sf::Vector2f& pt3);//NOLINT
   void draw(sf::RenderTarget& target, sf::RenderStates states) const;//NOLINT
   void setColorBlue();

  private:
   sf::VertexArray point;
 };

 #endif  // _HOME_VISOTHPEN_COMP2040_PS5_TRIANGLE_H_

```

Triangle.cpp

```
// Copyright Andy Pen
#include "Triangle.h"
#include <math.h>

Triangle::Triangle(const sf::Vector2f& pt1, const sf::Vector2f& pt2,
const sf::Vector2f& pt3) { //NOLINT
  point = sf::VertexArray(sf::Triangles, 3);
  point[0].position = pt1;
  point[1].position = pt2;
  point[2].position = pt3;
}

void Triangle::setColorBlue() {
  point[0].color = sf::Color::Blue;
  point[1].color = sf::Color::Blue;
  point[2].color = sf::Color::Blue;
}

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

```

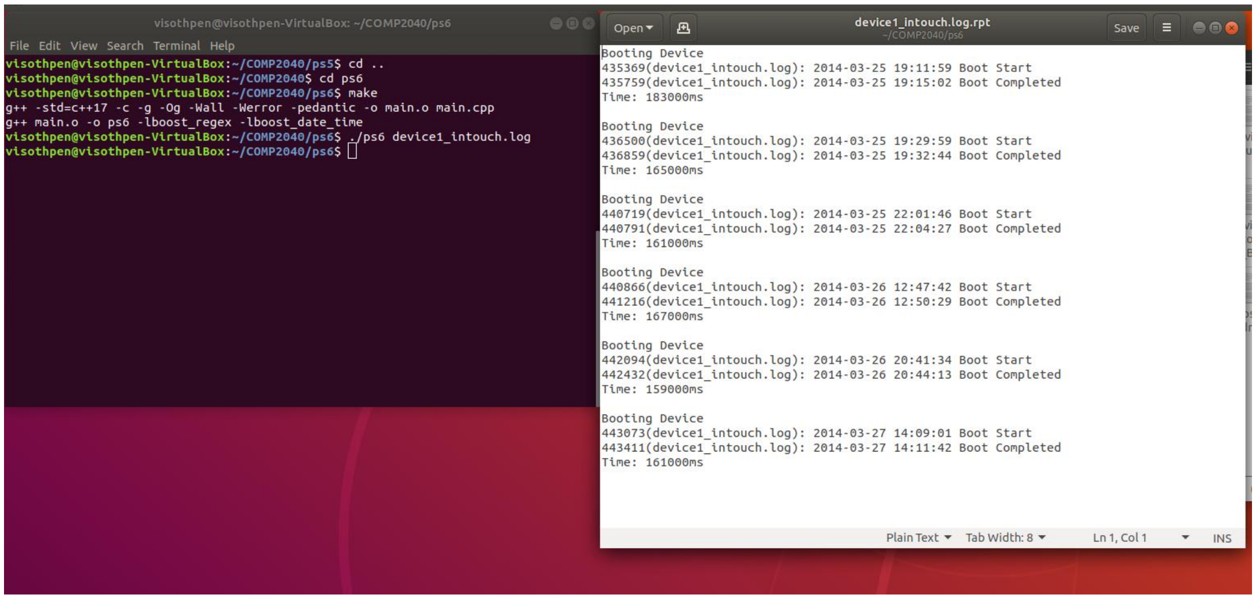
PS6: Kronos Time Clock – Intro to Regular Expression Parsing

Assignment
This assignment read in an entire InTouch log and report and create a text file report that chronologically describes each time the device was restarted.

Key Concepts
This assignment used regular expressions and the boost regex and time libraries to figure out the elapsed time between the server boot and boot completion. For this assignment, I didn't have any supporting header files or implementation files and was able to have everything in the main in which I used the boost libraries to regex_match and regex_search.

What I Learned
In this assignment, I learned how to use the boost regex library and the boost time library to figure out the times it took for each boot. I also learned how to link the boost libraries in the makefile and the basics of regular expressions.

Screenshot:


visothpen@visothpen-VirtualBox: ~/COMP2040/ps6
File Edit View Search Terminal Help
Open
device1_intouch.log.rpt
~/COMP2040/ps6
Save
Plain Text
Tab Width: 8
Ln 1, Col 1
INS

```
visothpen@visothpen-VirtualBox:~/COMP2040/ps5$ cd ..
visothpen@visothpen-VirtualBox:~/COMP2040$ cd ps6
visothpen@visothpen-VirtualBox:~/COMP2040/ps6$ make
g++ -std=c++17 -c -g -Og -Wall -Werror -pedantic -o main.o main.cpp
g++ main.o -o ps6 -lboost_regex -lboost_date_time
visothpen@visothpen-VirtualBox:~/COMP2040/ps6$ ./ps6 device1_intouch.log
visothpen@visothpen-VirtualBox:~/COMP2040/ps6$
```

```
Booting Device
435369(device1_intouch.log): 2014-03-25 19:11:59 Boot Start
435759(device1_intouch.log): 2014-03-25 19:15:02 Boot Completed
Time: 183000ms

Booting Device
436500(device1_intouch.log): 2014-03-25 19:29:59 Boot Start
436859(device1_intouch.log): 2014-03-25 19:32:44 Boot Completed
Time: 165000ms

Booting Device
440719(device1_intouch.log): 2014-03-25 22:01:46 Boot Start
440791(device1_intouch.log): 2014-03-25 22:04:27 Boot Completed
Time: 161000ms

Booting Device
440866(device1_intouch.log): 2014-03-26 12:47:42 Boot Start
441216(device1_intouch.log): 2014-03-26 12:50:29 Boot Completed
Time: 167000ms

Booting Device
442094(device1_intouch.log): 2014-03-26 20:41:34 Boot Start
442432(device1_intouch.log): 2014-03-26 20:44:13 Boot Completed
Time: 159000ms

Booting Device
443073(device1_intouch.log): 2014-03-27 14:09:01 Boot Start
443411(device1_intouch.log): 2014-03-27 14:11:42 Boot Completed
Time: 161000ms
```

Makefile:

```
CC = g++
CFLAGS = -std=c++17 -c -g -Og -Wall -Werror -pedantic
OBJ = main.o
DEPS = main.cpp
LIBS = -lboost_regex -lboost_date_time
EXE = ps6

all: $(OBJ)
  $(CC) $(OBJ) -o $(EXE) $(LIBS)
%.o: %.cpp $(DEPS)
        $(CC) $(CFLAGS) -o $@ $<

clean:
        rm $(OBJ) $(EXE)

```

main.cpp

```cpp
// Copyright Andy Pen 2021
#include <iostream>
#include <string>
#include <cstdlib>
#include <fstream>

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

 using boost::regex;
 using boost::gregorian::date;
 using boost::gregorian::from_simple_string;

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

 int main(int argc, char* argv[]) {
     if (argc != 2) {
         std::cout << "Error" << std::endl;
     }
     std::ifstream inputFile(argv[1], std::ifstream::in);
     if (!inputFile.is_open()) {
         std::cout << "Failed to open file. " << std::endl;
         return 0;
     }

     std::string fileName(std::string(argv[1]) + ".rpt");
     std::ofstream output;
     output.open(fileName.c_str());

     std::string s_date("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2}) ");
     std::string s_time("([0-9]{2}):([0-9]{2}):([0-9]{2})");
     std::string s_boot("(.*log.c.166.*)");
     std::string s_end("(.*oejs.AbstractConnector:Started
SelectChannelConnector.*)");//NOLINT
     boost::smatch match;

     regex r_boot(s_date + s_time + s_boot);
     regex r_end(s_date + s_time + s_end);

    // Check if boot is successful
     std::string s;
     int line;
     line = 1;
     bool state = false;
     ptime t1;
     ptime t2;

     while (getline(inputFile, s)) {
         if (regex_match(s, match, r_boot)) {
             if (state) {
                 output << "Boot Failed \n\n";
             }
       date d1(from_simple_string(match[0]));
       ptime temp(d1, time_duration(std::stoi(match[4]), std::stoi(match[5]), std::stoi(match[6])));//NOLINT
       t1 = temp;

       output << "Booting Device" << std::endl;
       output << line << "(" << argv[1] << "): ";
       output << match[1] << "-" << match[2] << "-" << match[3] << " ";
       output << match[4] << ":" << match[5] << ":" << match[6] << " ";
       output << "Boot Start" << std::endl;
       state = true;

     } else if (regex_match(s, match, r_end)) {
       if (state) {
         date d2(from_simple_string(match[0]));
         ptime temp(d2, time_duration(std::stoi(match[4]), std::stoi(match[5]), std::stoi(match[6])));//NOLINT
         t2 = temp;

         time_duration td;
         td = t2 - t1;

         output << line << "(" << argv[1] << "): ";
         output << match[1] << "-" << match[2] << "-" << match[3] << " ";
         output << match[4] << ":" << match[5] << ":" << match[6] << " ";
         output << "Boot Completed" << std::endl;

         output << "Time: ";
         output << td.total_milliseconds() << "ms \n\n";

         state = false;
       } else {
         output << "Unknown Boot\n\n";
       }
     }
     line++;
   }
   return 0;
 }

```