COMP.2040 Computing IV

Justin Nguyen

March 2022

Portfolio

Assignments

- PS0: Hello World with SFML
  - Getting started with SFML.
- PS1: Linear Feedback Shift Register + PhotoMagic
  - LFSR: A register (bit-array) with special methods for generating psuedo-random numbers.
  - Encrypting and decrypting photos using an LFSR.
- PS2: Static/Dynamic N-body Simulation
  - Generating a static universe
  - Simulating a moving universe with proper physics.
- PS3: Triangle Fractal (Sierpinski's Triangle)
  - Rendering a recursive triangle image.
- PS4: CircularBuffer + StringSound
  - Creating a circular buffer.
  - Using the CircularBuffer to simulate a guitar string being plucked.
- PS5: DNA Alignment
  - Most optimal alignment of two strings.
- PS6: RandomWriter
  - Using Markov chains to produce a generated text with probabilities of the frequencies characters to resemble the original work.
- PS7: Kronos Time Parsing
  - Introduction to Regex and Parsing a Kronos InTouch time clock log

PS0: Hello World with SFML

Description:

A tutorial meant to get us familiar with the SFML library. My computer was also set up to handle future projects. WSL2 was installed, and the SFML library. I then followed the most basic SFML tutorial of drawing a green circle on the screen.

The next task was to make it more unique, and make a sprite respond to keystrokes. I made my sprite, Robert Downey Jr., be able to move up and down on WASD. WS would cause the sprite to scale up or down, AD would cause the sprite to rotate left or right. Pressing E caused the sprite to reset to its initial state.

Discussion:

PS0 taught us how to use SFML. I got familiar with how to draw, position, and scale a sprite. SFML does make it easy to do all of this. Its classes are well written, and its methods are well named. For example, it is easy to infer that sf::Sprite::getPosition gives you the position of a sprite. If this was written the way C++ names their methods, it would probably look like sf::sp::pos.

In my own opinion, my code for controlling the sprite was gross looking.

```
// inside event loop...
if (sf::Keyboard::isKeyPressed(sf::Keyboard::E)) {  // reset
    rdj.setPosition(origPos);
    rdj.setRotation(origRot);
    rdj.setScale(origScale);
}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
    rdj.move(0, -1);
    rdj.scale(1+SCAL_RATE,1+SCAL_RATE);
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) { // vertical movement
    rdj.move(0, 1);
    rdj.scale(1-SCAL_RATE, 1-SCAL_RATE);
}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {//horizontal movement
    rdj.move(-1, 0);
    rdj.rotate(ROT_RATE);
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
    rdj.move(1, 0);
```

```
    rdj.rotate(-ROT_RATE);
}
```

which is messy. This was before I learned how to use lambdas in conjunction with a map. The map would be std::map<key, lambda>, where a key was binded to a lambda that executed some action.

Accomplished:

I accomplished making my first SFML project, and my first time ever making something of this sort.

Learned:

- How to use SFML and its libraries
- How to compile SFML libraries in a Makefile
- How to draw a sprite
  - Moving/positioning a sprite
  - Scaling/rotating a sprite

The complete basics were learned for interacting with SFML.

Key Algorithms/Data Structures/OO Designs:

None for this project.

Uncompleted/Not Working Features:

Everything for this project was completed, and works correctly.

Comments:

This was not my first time doing something like this. I used to make games when I was younger, so this felt pretty familiar having to bind certain functions to keys. Though, this assignment showed me how other systems represent ways of accepting user input, and I have to say SFML's is the worst. ROBLOX Studio has a very simple way of doing it, where it just uses a callback function and a class deals with the input. The user does not have to poll for key presses. It is done by using a callback function passed to an event handler.

```
-- luau
UserInputService.InputBegan:Connect(function(input, gameProcessedEvent)
    if (input.KeyCode == Enum.KeyCode.F) then
        print("I've pressed the F key!")
    end
end)
```

Obviously, this is much simpler. I wish SFML had followed suit.

Output: (evidence the code ran)

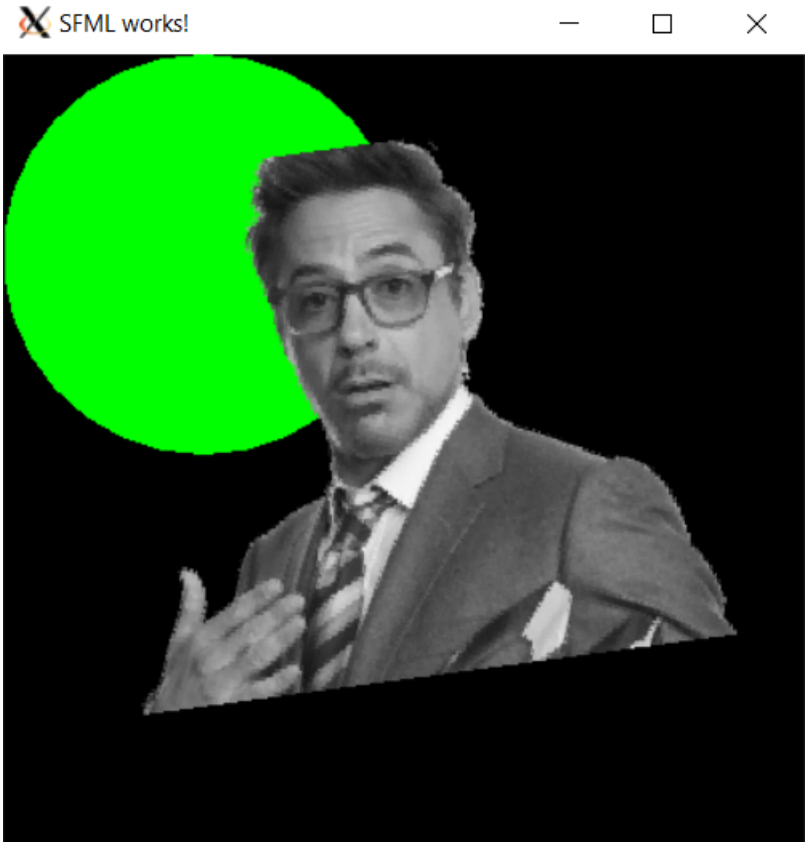


Figure 1: image

Makefile

```
EXE_NAME = sfmltutorial
OBJ = main.o
LIBS = -lsfml-system -lsfml-graphics -lsfml-window
TAR_NAME = PS0_JustinNguyen.tar.gz
CC = g++
FLAGS = -Wall -Werror -pedantic -std=c++17
#DEBUG = -g -ggdb3

all: $(EXE_NAME)

$(EXE_NAME): $(OBJ)
    $(CC) $(FLAGS) $(DEBUG) -o $@ $^ $(LIBS)

%.o: %.cpp %.hpp
    $(CC) $(FLAGS) $(DEBUG) -c $<

%.o: %.cpp
    $(CC) $(FLAGS) $(DEBUG) -c $^


run: # shortcut for you
    ./$(EXE_NAME)
dist:
    tar --exclude='.vscode' -czvf ../$(TAR_NAME) .
    mv ../$(TAR_NAME) .
clean:
    rm -f $(OBJ)
distclean: clean
    rm -f $(EXE_NAME) $(TAR_NAME)
```

main.cpp

```
/**
    Name: Justin Nguyen

    Course: COMP IV
    Instructor: Daly
    Assignment: PS0

    Date: 1/22/2022
    Due: 1/24/2022

    Description: Sprite of Robert Downey Jr., press W or S to move and scale the image up o
    or A and D to move and rotate the image left and right.
```

```
13      Press E to reset to original orientation, size, and position.
14
15      Problems: Occasionally, RDJ's rendering will flash rapidly while moving out of the windo
16      Currently, I am unsure of how to fix this.
17  **/
18
19  #include <SFML/Graphics.hpp>
20  #include <SFML/Graphics/Text.hpp>
21  #include <iostream>
22
23  const double ROT_RATE = 0.25;
24  const double SCAL_RATE = 0.01;
25
26  int main(void) {
27      sf::RenderWindow window(sf::VideoMode(400, 400), "SFML works!");
28
29      sf::CircleShape shape(100.f);
30
31      sf::Texture texture;
32      sf::Sprite rdj;
33      sf::Text test;
34
35      //original position, orientation, sclae
36      sf::Vector2f origPos, origScale;
37      double origRot;
38
39      if (!texture.loadFromFile("sprite.png"))
40          return 1;
41
42      shape.setFillColor(sf::Color::Green);
43
44      rdj.setTexture(texture);
45      origPos = rdj.getPosition();
46      origRot = rdj.getRotation();
47      origScale = rdj.getScale();
48
49      while (window.isOpen()) {
50          sf::Event event;
51          while (window.pollEvent(event)) {
52              if (event.type == sf::Event::Closed)
53                  window.close();
54          }
55
56          window.clear();
57
58          if (sf::Keyboard::isKeyPressed(sf::Keyboard::E)) {
```

```
59              rdj.setPosition(origPos);
60              rdj.setRotation(origRot);
61              rdj.setScale(origScale);
62          }
63
64          if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
65              rdj.move(0, -1);
66              rdj.scale(1+SCAL_RATE,1+SCAL_RATE);
67          }
68          if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) { // vertical movement
69              rdj.move(0, 1);
70              rdj.scale(1-SCAL_RATE, 1-SCAL_RATE);
71          }
72
73          if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {//horizontal movement
74              rdj.move(-1, 0);
75              rdj.rotate(ROT_RATE);
76          }
77          if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
78              rdj.move(1, 0);
79              rdj.rotate(-ROT_RATE);
80          }
81
82          if (rdj.getPosition().x + rdj.getScale().x > window.getSize().x)
83              rdj.setPosition(0,rdj.getPosition().y);
84
85          if (rdj.getPosition().x + rdj.getScale().x < 0)
86              rdj.setPosition(window.getSize().x,rdj.getPosition().y);
87
88          if (rdj.getPosition().y + rdj.getScale().y > window.getSize().y)
89              rdj.setPosition(rdj.getPosition().x, 0);
90
91          if (rdj.getPosition().y + rdj.getScale().y < 0)
92              rdj.setPosition(rdj.getPosition().x, window.getSize().y);
93
94          window.draw(shape);
95          window.draw(rdj);
96
97          window.display();
98      }
99
100     return 0;
101 }
```

PS1: Linear Feedback Shift Register & PhotoMagic

Description:

PS1a: LFSR

Creating an LFSR (Linear Feedback Shift Register) to generate psuedo-random numbers. An LFSR is an array of bits. What it is mainly used for is to generate psuedo-random numbers. How it generates a psuedo-random number is by XORing some bits at pre-defined tap positions with the bit that got shifted off, then right-shifting the entire register.

Three important factors are what causes it the quality of the psuedo-random numbers it produces are: - Number of bits (size of the register) - Intial seed - Tap positions

This project was then unit-tested to ensure the given inputs matched to an expected output by rigorously testing the FibLFSR class.

PS1b: PhotoMagic

Encrypting and decrypting an image using a FibLFSR. By using a FibLFSR, if you generate a random number, then XOR it with the [R,G,B], you can encrypt an image. The image appears as noise, with its pixels looking like the static on a TV screen. By performing the same operation on the output image, it can be reversed to get the input image.

Discussion:

To first create the 16-bit FibLFSR, I used what instantly came to mind: std::bitset<16>. Which is already half the project done. std::bitset::operator>>(), which would right shift its representation, and std::bitset::operator[] which would give you a reference to the bit at that index. Really, the project was almost done from the start. All I had to do left was XOR the falling bit at the tap positions.

The next part was the unit testing. For this unit test, as we were given what the output should look like from a given input, I wrote my tests in a similar manner to see if it would match those outputs. The tests did match the outputs.

Finally, using SFML and my FibLFSR class, I encrypted an image. The encryption was quite simple: take a pixel from the image and XOR it with a randomlmy generated number

Accomplished

- Creating a psuedo-random number generator, and learned how to thoroughly unit-test a program.
- Encrypting and decrypting an image with my FibLFSR class.

Learned:

- One of the many representations of how to make a psuedo-random number generator.
- Usage of std::bitset
- The BOOST library, and its unit testing functions.
- How to apply something I made earlier to do another task. In this case, using our FibLFSR to encrypt and decrypt an image.
- One way how images are encrypted and decrypted
  - By XORing its pixels with a psuedo-randomly generated number, the image can be encrypted.
  - By doing the same operation on the output image, the input image can be recovered.

Key Algorithms/Data Structures/OO Designs:

std::bitset<>

This data structure incredibly useful for this project. This data structure is a bit array, which is an array of 0s and 1s. It had operations such as left/right shift, outputting the internal representation as a binary string or converting the representation to an unsigned int.

Making my own bit array, which is usually done with just an unsigned int, would be incredibly tedious and buggy. I thank the designers of the C++ standard library every day.

FibLFSR

Of course, we must discuss the KEY KEY data structure, the FibLFSR. Without this, the image could not have been encrypted/decrypted. This structure is basically a 16-bit register. The operations performed on it can generate psuedo-random numbers.

Uncompleted/Not Working Features:

Everything for this project was completed, and works correctly.

Output: (evidence the code ran)

Figure 2: *Output from unit-test*

```
./ps1a
Running 1 test case...

*** No errors detected
```

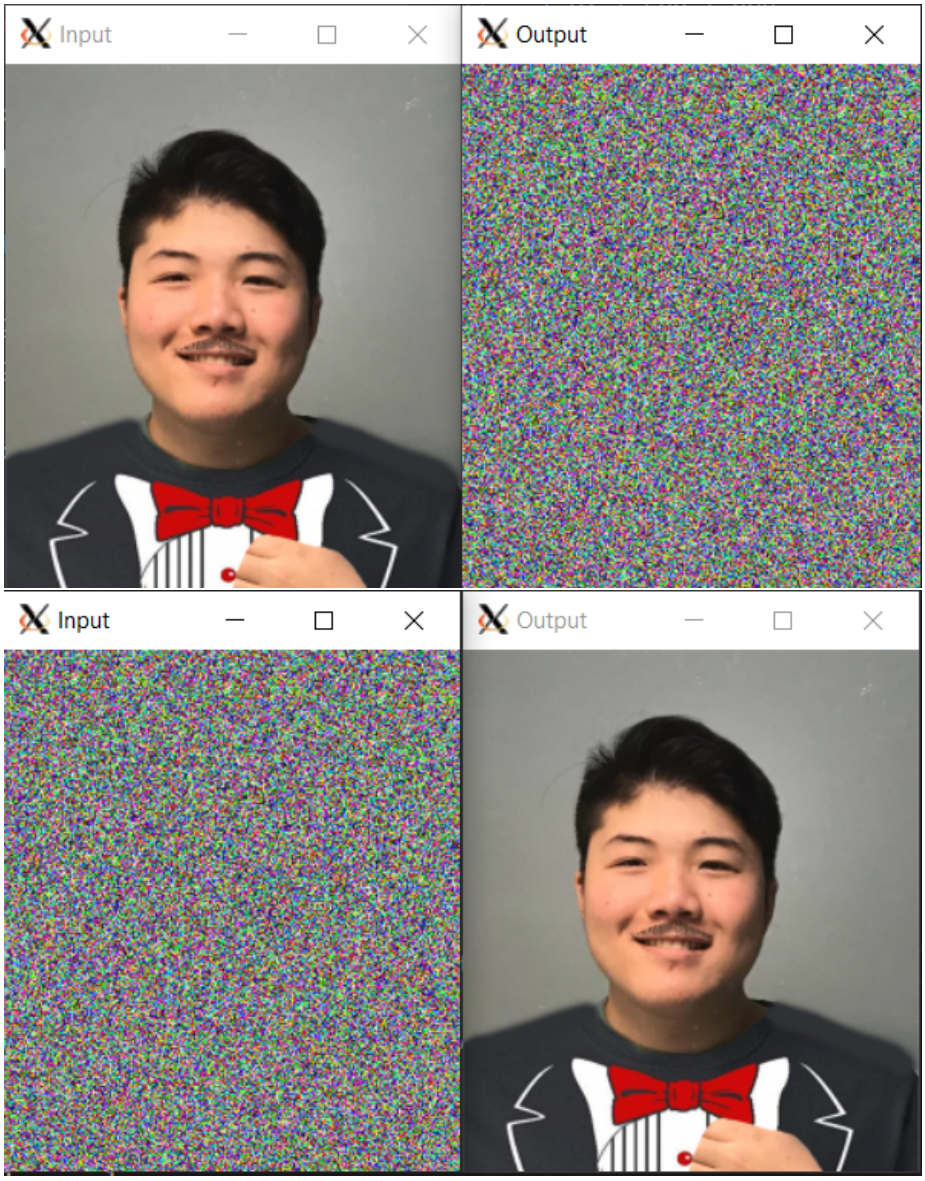


Figure 3: *Encoding input image, and decoding output image.*

Makefile

```
EXE_NAME = psx
OBJ = test.o FibLFSR.o
BOOST = -lboost_unit_test_framework
TAR_NAME = PS1a_JustinNguyen.tar.gz
CC = g++
FLAGS = -Wall -Werror -pedantic -std=c++14 #-g # debug flag

$(EXE_NAME): $(OBJ)
    $(CC) $(FLAGS) -o $@ $^ $(BOOST)

%.o: %.cpp %.h
    $(CC) $(FLAGS) -c $<

%.o: %.cpp
    $(CC) $(FLAGS) -c $^

all: $(EXE_NAME)

lint:
    clear
    cpplint *.*pp

run: # this is easier to type than ./exe_name
    ./$(EXE_NAME)
dist: # extra instruction to move it back into wd. earlier i tried to do it in the wd, but t
    tar --exclude='.vscode' -czvf ../$(TAR_NAME) .
    mv ../$(TAR_NAME) .
clean:
    rm -f $(OBJ)
distclean: clean
    rm -f $(EXE_NAME) $(TAR_NAME)
```

main.cpp

```cpp
#include <string>
#include <iostream>

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

#include "FibLFSR.hpp"

using std::cout;
using std::endl;

void transform(sf::Image&, FibLFSR&);

int main(int argc, char** argv) {
    if (argc < 4) {
        std::cerr << "ERROR: Invalid number of arguments!" << endl;
        return 1;
    }

    sf::Image iimg, oimg;
    sf::Texture itexture, otexture;
    sf::Sprite isprite, osprite;
    std::string iname(argv[1]), oname(argv[2]), seed(argv[3]);
    FibLFSR lfsr(seed);

    if (!iimg.loadFromFile(iname)) {
        std::cerr << "ERROR - Missing file: " << iname << endl;
        return 1;
    }

    oimg = iimg;  // copy input into output

    sf::RenderWindow window1(sf::VideoMode(iimg.getSize().x, iimg.getSize().y), "Input");
    sf::RenderWindow window2(sf::VideoMode(oimg.getSize().x, oimg.getSize().y), "Output");

    transform(oimg, lfsr);

    itexture.loadFromImage(iimg);
    isprite.setTexture(itexture);

    otexture.loadFromImage(oimg);
    osprite.setTexture(otexture);

```

```
45      while (window1.isOpen() && window2.isOpen()) {
46          sf::Event event;
47          while (window1.pollEvent(event)) {
48              if (event.type == sf::Event::Closed)
49                  window1.close();
50          }
51          while (window2.pollEvent(event)) {
52              if (event.type == sf::Event::Closed)
53                  window2.close();
54          }
55          window1.clear();
56          window1.draw(isprite);
57          window1.display();
58
59          window2.clear();
60          window2.draw(osprite);
61          window2.display();
62      }
63
64      if (!oimg.saveToFile(oname)) {
65          std::cerr << "ERROR - Could not save to file: " << oname << std::endl;
66          return 1;
67      }
68
69      return 0;
70  }
71
72  void transform(sf::Image& target, FibLFSR& prng) {
73      const int k = 8;  // larger k takes longer time, requires optimizations
74      sf::Color pixelAt;
75      for (unsigned int x = 0; x < target.getSize().x; ++x) {
76          for (unsigned int y = 0; y < target.getSize().y; ++y) {
77              pixelAt = target.getPixel(x, y);
78              unsigned int r = pixelAt.r, g = pixelAt.g, b = pixelAt.b;
79              target.setPixel(x, y, sf::Color(r^prng.generate(k), g^prng.generate(k), b^prng.g
80          }
81      }
82  }
```

test.cpp

```cpp
// Copyright 2022 Justin Nguyen
#include <iostream>
#include <string>

#include "FibLFSR.hpp"

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

using boost::test_tools::output_test_stream;

BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
    std::string seed("1011011000110110");
    FibLFSR l(seed);
    FibLFSR l2 = l;
    FibLFSR l3 = l;
    output_test_stream output;

    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);

    BOOST_REQUIRE(l2.generate(9) == 51);
    // MY TEST CASES:
    // from the pdf, testing known inputs and expecting known outputs
    // testing generate(5) and seeing if it
    // matches up with assignment pdf values
    BOOST_REQUIRE(l3.generate(5) == 3);
    BOOST_REQUIRE(l3.generate(5) == 6);
    BOOST_REQUIRE(l3.generate(5) == 14);

    // testing output function to see if it matches with expected value.
    // consulted boost documentation to learn how to do this

    output << l3;
    BOOST_TEST(output.is_equal("0000110011001110"));
}
```

FibLFSR.hpp

```cpp
// Copyright 2022 Justin Nguyen
#ifndef _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS1A_FIBLFSR_HPP_
#define _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS1A_FIBLFSR_HPP_

#include <string>
#include <bitset>
#include <iostream>

class FibLFSR {
 public:
    explicit FibLFSR(std::string seed): bitArray(seed) {}
    int step();
    int generate(int k);

    friend std::ostream& operator<<(std::ostream& out, const FibLFSR& obj) {
        return out << obj.bitArray.to_string();
    }

 private:
    std::bitset<16> bitArray;  // bit array
};

#endif  // _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS1A_FIBLFSR_HPP_
```

FibLFSR.cpp

```cpp

// Copyright 2022 Justin Nguyen
#include "FibLFSR.hpp"
#include <bitset>
#include <string>

static const int numTaps = 3;
static const int taps[numTaps] = {13, 12, 10};

int FibLFSR::step() {
    int msb = bitArray[bitArray.size()-1];  // save the most significant bit
    for (int i = 0; i < numTaps; ++i)
        msb ^= bitArray[taps[i]];   // xoring msb at tap positions
    bitArray <<= 1;
    return bitArray[0] = msb;
}

int FibLFSR::generate(int k) {
```

```
19      std::string bits;
20      for (int i = 0; i < k; i++)
21          bits += std::to_string(this->step());  // binary string
22      return std::bitset<32>(bits).to_ulong();
23  }
24
25
26  /* we code golfin
27  int FibLFSR::step() {
28      int msb = bitArray[15]^13^12^10;//how to write bad code
29      bitArray <<= 1;
30      return bitArray[0] = msb;
31  }
32  */
```

PS2: Generating a Static & Dynamic N-body Simulation

Description:

Using SFML, a static representation of our solar system was created. With the sun as the center, and the nine planets as its celestial bodies.

Later, in the next project, using some physics concepts and laws, a dynamic simulation was created by causing the static one to move.

Discussion:

This project is my 2nd favorite so far in the class. For the design of my project, I first built it from the ground up, so bottom-up. I built the CelestialBody class first. CelestialBodies should manage their own physics, while the universe puts them into play. I had this thought in mind, so my CelestialBody set its new position after calculating it, and the Universe put it into play by stepping. This approach kept my code cleaned and organized.

For designing the Universe class, I actually did something a little different from others. I had Universe inherit from RenderWindow. I felt this made more sense than Universe inheriting from sf::Drawable for a couple of reasons. If the Universe contained the planets, why not have it also draw the planets. I overloaded sf::RenderWindow::draw on Universe to just be Universe::draw, and all it does is draw its internal CelestialBodies. It felt more natural this way instead of making the Universe an sf::Drawable.

Universe would call CelestialBody::GetSetUpdatedPosition, which would set its internal position, then Universe would set the CelestialBody's real position.

Accomplished:

I accomplished putting together a static solar system. This was also the first time I've ever used an std::unique_ptr, which is incredibly useful. That let me make my program more memory safe, so it did not leak memory and I did not have to explcitly clean up a CelestialBody class. Also, as there can only be one planet of this type, std::unique_ptr was the best memory container for this job.

After making a static system, it was modified to now be dynamic. Using some physics concepts and equations, the planets began to orbit counterclockwise around their sun. This part was hard, yet I found to be amazing. Watching something I made come to life is a feeling you don't get often.

Learned:

- How to use std::unique_ptr<>
- How to position sprites properly

- How to parse a data file more accurately
- How to use composition
  - How to have two classes interact with each other in an elegant way.
- How to use physics to move CelestialBodies
- How to scale universe coordinates to window coordinates
- How to solve the whitebox problem
  - Required the usage of dynamically allocating CelestialBody, so that it would hold onto its original contents.

Key Algorithms/Data Structures/OO Designs:

- It was not explicitly stated, but OO composition was a part of this project. We used one class inside another, or composed one class with another. It would make the Universe class bloated if the Universe created its own planet and methods to go along with, but instead the organized approach of just having a separate class to represent a celestial body.

Uncompleted/Not Working Features:

Everything for this project was completed, and works.

Output: (evidence the code ran)

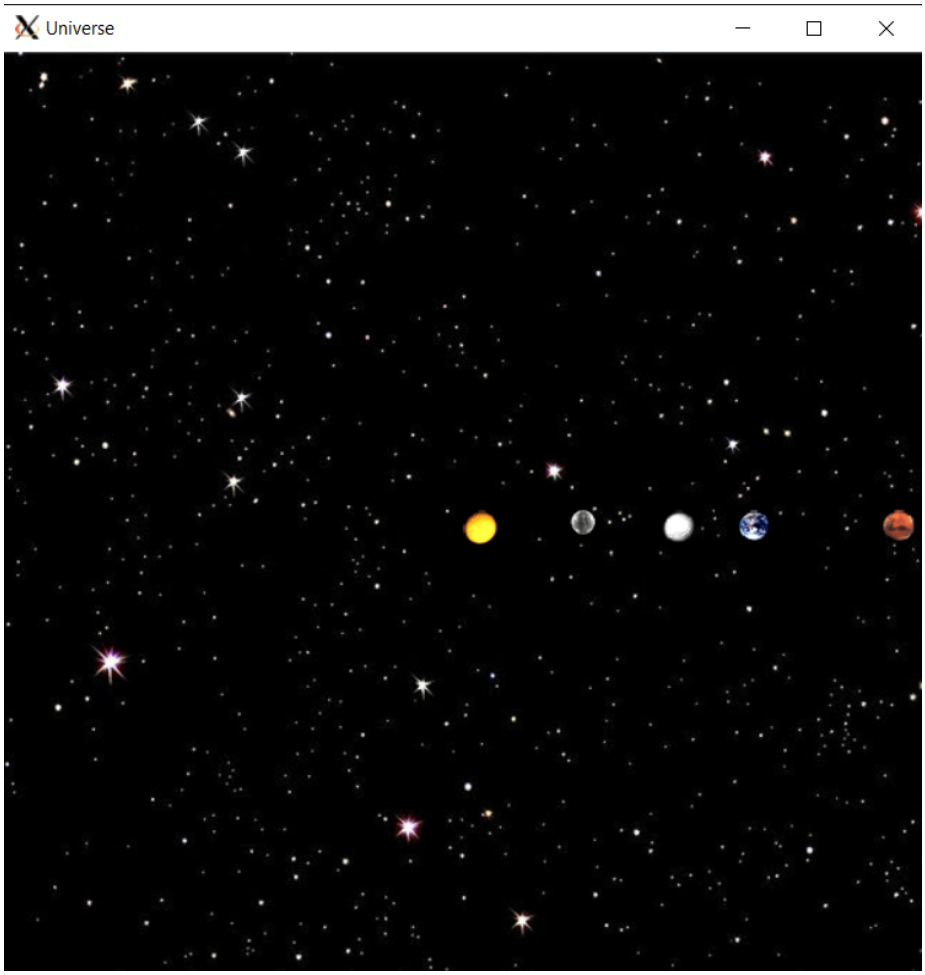


Figure 4: *Static N-Body Simulation of our Solar System*

Makefile

```
EXE_NAME = Nbody
OBJ = CelestialBody.o Universe.o main.o
LIBS = -lsfml-system -lsfml-graphics -lsfml-window  #-lboost_unit_test_framework
TAR_NAME = PS2A_JustinNguyen.tar.gz
CC = g++
FLAGS = -Wall -Werror -pedantic -std=c++14 #-ggdb3 #-g # debug flag

all: $(EXE_NAME)

$(EXE_NAME): $(OBJ)
    $(CC) $(FLAGS) -o $@ $^ $(LIBS)

%.o: %.cpp %.hpp
    $(CC) $(FLAGS) -c $<

%.o: %.cpp
    $(CC) $(FLAGS) -c $^


run: # shortcut for you
    ./$(EXE_NAME) < planets.txt
dist:
    tar --exclude='.vscode' -czvf ../$(TAR_NAME) .
    mv ../$(TAR_NAME) .
clean:
    rm -f $(OBJ)
distclean: clean
    rm -f $(EXE_NAME) $(TAR_NAME)
```

main.cpp

```cpp
#include <iostream>
#include <string>

#include <SFML/Graphics/Text.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>

#include "Universe.hpp"

const unsigned int VPSIZE(512);
const double UNIVERSE_RADIUS(1e10);

int main(int argc, const char** const argv) {
    if (argc != 3) {
        std::cerr << "INVALID NUMBER OF ARGUMENTS! PLEASE SPECIFY TIME AND DELTA TIME IN THA
        return 1;
    }
    double t = std::stod(argv[1]), dt = std::stod(argv[2]);
    // double t = 157788000.0, dt = 25000.0; // fast test
    if (dt >= t) {
        std::cerr << "INVALID ARGUMENTS! DELTA TIME CANNOT BE MORE THAN TOTAL TIME." << std
        return 1;
    }

    Universe u(t, VPSIZE, UNIVERSE_RADIUS, "assets/starry_sky.jpg");
    sf::Font font;
    sf::Text elaspedTime(std::string(std::to_string(u.GetElaspedTime())), font);
    sf::Music song;

    song.openFromFile("assets/beep.ogg"); // things go horribly wrong when i try this, i hop
    song.play();

    elaspedTime.setCharacterSize(24);
    elaspedTime.setFillColor(sf::Color::Red);
    elaspedTime.setStyle(sf::Text::Regular);

    std::cin >> u;

    u.Begin();

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

```
45
46          u.sf::RenderWindow::draw(elaspedTime);
47          u.draw();
48
49          u.step(dt);
50
51          u.display();
52          elaspedTime.setString(std::string(std::to_string(u.GetElaspedTime())));
53      }
54      std::cout << u << std::endl;
55      return 0;
56  }
```

Universe.hpp

```
#ifndef UNIVERSE_HPP
#define UNIVERSE_HPP

#include <vector>
#include <memory>
#include <iostream>

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

#include "CelestialBody.hpp"

using sf::Vector2f;

const static string UNAME("Universe");

class Universe: public sf::RenderWindow {
    public:
        Universe(double windowSize = 600, double radius = 1e10, std::string filename = "")
        : sf::RenderWindow(sf::VideoMode(windowSize,windowSize), UNAME),  vpd(windowSize), r
            SetBackgroundImage(filename);
        };
        Universe(sf::Vector2u dimensions, double radius = 1e10, std::string filename = "")
        : sf::RenderWindow(sf::VideoMode(dimensions.x, dimensions.y), UNAME), r(radius), fn(
            SetBackgroundImage(filename);
        };
        void draw(void);
        void SetBackgroundImage(std::string filename);
        friend std::istream& operator>>(std::istream&, Universe&);
    private:
        Vector2f ToWindowCoords(Vector2f) const;
        double utw(double) const;
        std::vector<std::unique_ptr<CelestialBody>> cbs;
        double vpd; // viewport dimension
        double r; // universe radius
        std::string fn;
        sf::Texture texture;
        sf::Sprite bgImg;
};

#endif
```

Universe.cpp

```
#include <string>
#include <memory>
#include <iostream>

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

#include "Universe.hpp"

using sf::Vector2f;

double Universe::utw(double ucoord) const {
    return (r + ucoord)*vpd/(2*r);
}

Vector2f Universe::ToWindowCoords(Vector2f ucoord) const {
    return Vector2f(utw(ucoord.x),utw(-ucoord.y));
}

void Universe::SetBackgroundImage(std::string filename) {
    fn = filename;
    texture.loadFromFile(filename);
    bgImg.setTexture(texture);
}

void Universe::draw(void) {
    if (!fn.empty())
        sf::RenderWindow::draw(bgImg);
    for (auto& cb: cbs) {
        cb->sprite.setPosition(ToWindowCoords(cb->p) + cb->GetSpriteOffset());
        sf::RenderWindow::draw(*cb);
    }
}

std::istream& operator>>(std::istream& is, Universe& u) {
    unsigned int lineCount;
    double uRadius;
    std::string uRadius_str;

    is >> lineCount;
    is.ignore(80, '\n');
    is >> uRadius_str;
    uRadius = std::stod(uRadius_str);

```

```
45      for (unsigned int i = 0; i < lineCount; i++) {
46          std::unique_ptr<CelestialBody> newBody(new CelestialBody);
47          is >> *newBody;
48          u.cbs.push_back(std::move(newBody));
49      }
50      u.r = uRadius;
51      u.setTitle(UNAME);
52      return is;
53  }
```

CelestialBody.hpp

```
#ifndef CELESTIALBODY_HPP
#define CELESTIALBODY_HPP

#include <SFML/Graphics.hpp>
#include <iostream>

using sf::Vector2f;
using std::string;

class CelestialBody: public sf::Drawable {
    public:
        CelestialBody(void) {};
        CelestialBody(Vector2f position, Vector2f velocity, double mass, string filename);

        Vector2f GetPosition(void) const {return p;};
        Vector2f GetVelocity(void) const {return v;};
        double GetMass(void) const {return m;};
        friend class Universe;
        friend std::istream& operator>>(std::istream&, CelestialBody&);

    private:
        virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
        Vector2f GetSpriteOffset() const;
        sf::Vector2f p, v;
        double m;
        sf::Texture texture;
        sf::Sprite sprite;
};

#endif
```

CelestialBody.cpp

```

#include <iostream>
#include <string>

#include <SFML/Graphics.hpp>

#include "CelestialBody.hpp"
#include "Physics.hpp"

sf::Vector2f CelestialBody::GetSpriteOffset() const {
    return Vector2f(sprite.getScale().x/2, -sprite.getScale().y/2);
```

```
12  }
13
14  CelestialBody::CelestialBody(Vector2f position, Vector2f velocity, double mass, string filen
15  : p(position), v(velocity), m(mass), fn(filename){
16      texture.loadFromFile(filename);
17      sprite.setTexture(texture);
18  }
19
20  void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states) const{
21      target.draw(sprite, states);
22  }
23
24  sf::Vector2f CelestialBody::GetSetUpdatedPosition(const std::vector<std::unique_ptr<Celestia
25      sf::Vector2f totalForce(0,0);
26      for (auto& cb: cbs) {
27          if (cb.get() == this) {
28              continue;
29          }
30          totalForce += GetForceBetween(*this, *cb); // getting net force on this body from al
31      }
32      v += sf::Vector2f(dt*totalForce.x/m, dt*totalForce.y/m);
33      return p = GetPosition() + sf::Vector2f(-v.x*dt, -v.y*dt);
34  }
35
36  std::istream& operator>>(std::istream& is, CelestialBody& cb) {
37      std::string vals[5];
38
39      for (unsigned int i = 0; i < 5; i++) {
40          is.ignore(80, ' ');
41          is >> vals[i];
42      }
43      is.ignore(80, ' ');
44      is >> cb.fn;
45
46      cb.texture.loadFromFile("assets/"+cb.fn);
47      cb.sprite.setTexture(cb.texture);
48
49      cb.p = Vector2f(std::stod(vals[0]),std::stod(vals[1]));
50      cb.v = Vector2f(std::stod(vals[2]),std::stod(vals[3]));
51      cb.m = std::stod(vals[4]);
52
53      return is;
54  }
```

Physics.hpp

```
#ifndef PHYSICS_HPP
#define PHYSICS_HPP

#include "CelestialBody.hpp"

namespace P_CONSTANTS {
    const double G(6.674*(1/1e11)); // gravitational constant
}

using namespace P_CONSTANTS;

double GetMagnitude(const sf::Vector2f, const sf::Vector2f);

double GetForce(const CelestialBody&, const CelestialBody&);

sf::Vector2f GetForceBetween(const CelestialBody&, const CelestialBody&);

#endif
```

Physics.cpp

```
#include <cmath>

#include <SFML/Graphics.hpp>

#include "Physics.hpp"

using namespace P_CONSTANTS;

double GetMagnitude(const sf::Vector2f a, const sf::Vector2f b) {
    sf::Vector2f c(a-b);
    return sqrt(c.x*c.x + c.y*c.y);
}

double GetForce(const CelestialBody& a, const CelestialBody& b) {
    double r = GetMagnitude(a.GetPosition(),b.GetPosition());
    return G*a.GetMass()*b.GetMass()/(r*r); // F = G * (Mm) / r^2
}

sf::Vector2f GetForceBetween(const CelestialBody& a, const CelestialBody& b) {
    double F = GetForce(a,b);
    double r = GetMagnitude(a.GetPosition(),b.GetPosition());
    sf::Vector2f delta(a.GetPosition()-b.GetPosition());
    return sf::Vector2f(F*delta.x/r, F*delta.y/r);
```

```
24  }
```

PS3: Triangle Fractal (Sierpinski's Triangle)

Description:

Sierpinski's triangle is a triangle with triangles on its vertices with triangles on the vertices of the earlier triangles with... In summary, it is a recursively generated fractal image.

This project was the first time I've ever used a linter, so the code for this project follows Google's style.

Discussion:

This project made me hate SFML. SFML, for some odd reason, does not have a triangle primitive. Instead, they recommended you to use sf::Circle(radius, 3) to draw a triangle. Or a circle with 3 edges. Their reasoning:

- There's no dedicated class for regular polygons, in fact you can represent a regular polygon with any number of sides using the sf::CircleShape class: Since circles are approximated by polygons with many sides, you just have to play with the number of sides to get the desired polygons. A sf::CircleShape with 3 points is a triangle, with 4 points it's a square, etc.

Which sounds OK in theory, but is absolutely terrible in practice. When I attempted to size the triangle according to the input first input parameter given in argv, it did not correctly match. How I knew? I actually did not, but I realized how the hell would I set the side length of this triangle if I could only set the radius parameter in sf::Circle. Which made me solve for this problem:

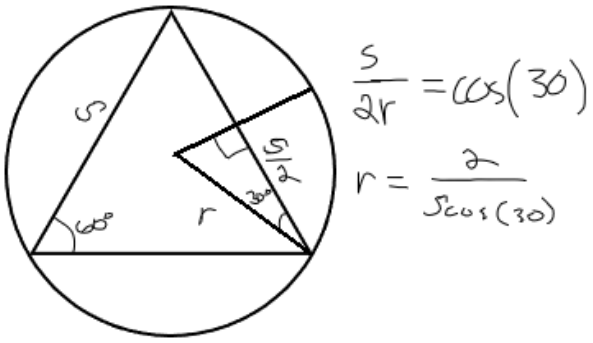


Figure 5: *A fun, but useless problem.* As I later found out, the math in the image is wrong, and r is actually `r = s/(2 * cos(30))`.

In the end, I went with sf::ConvexShape, and made this beauty to draw my triangle:

```
Triangle::Triangle(double sidelength, Vector2f pos)
:trigon(NUM_TRI_SIDES), c(sidelength) {
    trigon.setPoint(0, pos);  //  left most
    trigon.setPoint(1, pos + Vector2f(c, 0));  //  right most
    trigon.setPoint(2, pos + Vector2f(c/2, c));  //  bottom most
}
```

Elegantly simple just like the equilateral triangle.

Designing the rest of the project was not too hard. The main guts of TFractal comes from its recursive function:

```
void ftree(const Triangle& triangle, unsigned int depth) {
    if (depth == 0)
        return;
    double SL = triangle.GetSideLength(3)/2.f;
    Triangle
    sub1(SL, triangle.GetPointPosition(0) + Vector2f(-SL/2, -SL)),  //  leftmost
    sub2(SL, triangle.GetPointPosition(1)),  //  rightmost
    sub3(SL, triangle.GetPointPosition(2) + Vector2f(-SL, 0));  //  bottommost

    triangles.push_back(sub1);
    triangles.push_back(sub2);
    triangles.push_back(sub3);

    ftree(sub1, depth - 1);
    ftree(sub2, depth - 1);
    ftree(sub3, depth - 1);
}
```

Something about recursion is just so elegant, and writing this function felt really satisfying.

Accomplished:

Making a beautifully generated recursive fractal design.

Learned:

- How to solve for a circle's radius given the length of a side of an inscribed equilateral triangle.
- How to make your own sf::ConvexShape
  - How to override sf::ConvexShape::draw
- How to use a linter (cpplint)

Key Algorithms/Data Structures/OO Designs:

None for this project.

Uncompleted/Not Working Features:

Everything for this project was completed, and works.

Output: (evidence the code ran)

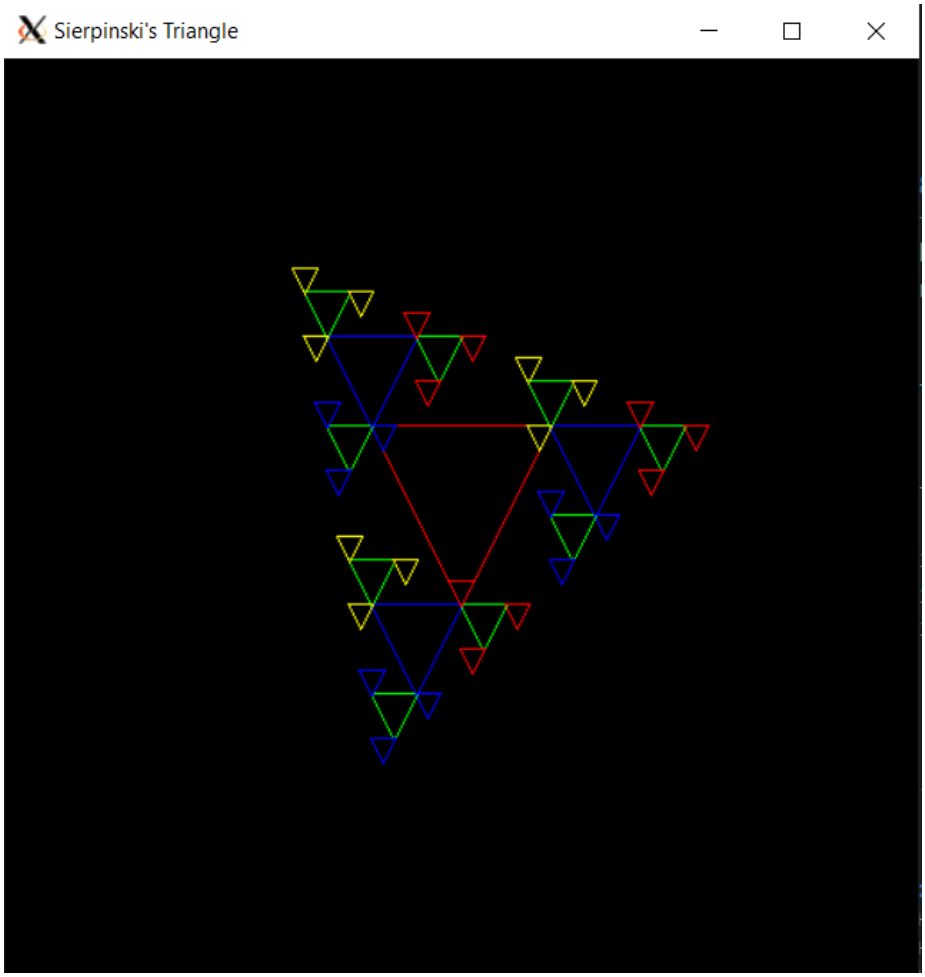


Figure 5: *A triangle fractal with recursion depth 3.*

Makefile

```
EXE_NAME = TFractal
OBJ = Triangle.o TFractal.o
LIBS = -lsfml-system -lsfml-graphics -lsfml-window
TAR_NAME = PS3_JustinNguyen.tar.gz
CC = g++
FLAGS = -Wall -Werror -pedantic -std=c++17

LL = cpplint
LFLAGS = --filter=-runtime/references
#DEBUG = -g -ggdb3

all: $(EXE_NAME)

$(EXE_NAME): $(OBJ)
    $(CC) $(FLAGS) $(DEBUG) -o $@ $^ $(LIBS)

%.o: %.cpp %.hpp
    $(CC) $(FLAGS) $(DEBUG) -c $<


%.o: %.cpp
    $(CC) $(FLAGS) $(DEBUG) -c $^

run: # shortcut for you
    ./$(EXE_NAME) 100 3
lint:
    cpplint *.*pp

dist:
    tar --exclude='.vscode' -czvf ../$(TAR_NAME) .
    mv ../$(TAR_NAME) .
clean:
    rm -f $(OBJ)
distclean: clean
    rm -f $(EXE_NAME) $(TAR_NAME)
```

main.cpp

```cpp
// Copyright 2022 <Justin Nguyen>
#include <iostream>
#include <string>
#include <vector>

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

#include "Triangle.hpp"

using sf::Vector2f;

const unsigned int WINDOW_SIZE(512);

const size_t NUM_COLORS(4);
size_t colorAt(0);

std::vector<Triangle> triangles;

const sf::Color colors[NUM_COLORS] = {
    sf::Color::Red,
    sf::Color::Blue,
    sf::Color::Green,
    sf::Color::Yellow
};

void setAppearance(Triangle& triangle); // NOLINT
void ftree(const Triangle& triangle, unsigned int depth);

int main(int argc, char** argv) {
    if (argc < 3) {
        std::cerr << "INVALID NUMBER OF ARGUMENTS" << std::endl;
        return 1;
    }
    unsigned int L(std::stoi(argv[1])), N(std::stoi(argv[2]));

    sf::RenderWindow window(
        sf::VideoMode(WINDOW_SIZE, WINDOW_SIZE), "Sierpinski's Triangle");

    Triangle first(L, sf::Vector2f(WINDOW_SIZE/2 - L/2, WINDOW_SIZE/2 - L/2));

    setAppearance(first);
    colorAt++;

```

```
45      triangles.push_back(first);
46
47      ftree(first, N);
48
49      while (window.isOpen()) {
50          sf::Event event;
51          while (window.pollEvent(event)) {
52              if (event.type == sf::Event::EventType::Closed)
53                  window.close();
54          }
55          window.clear();
56          for (const auto& t : triangles)
57              window.draw(t);
58          window.display();
59      }
60
61      return 0;
62  }
63
64  void setAppearance(Triangle& triangle) { // NOLINT
65      triangle.SetFillColor(sf::Color::Transparent);
66      triangle.SetOutlineWidth(1);
67      triangle.SetOutlineColor(colors[colorAt % NUM_COLORS]);
68  }
69
70  void ftree(const Triangle& triangle, unsigned int depth) {
71      if (depth == 0)
72          return;
73      double SL = triangle.GetSideLength(3)/2.f;
74      Triangle
75      sub1(SL, triangle.GetPointPosition(0) + Vector2f(-SL/2, -SL)),  //  leftmost
76      sub2(SL, triangle.GetPointPosition(1)),  //  rightmost
77      sub3(SL, triangle.GetPointPosition(2) + Vector2f(-SL, 0));  //  bottommost
78
79      setAppearance(sub1);
80      setAppearance(sub2);
81      setAppearance(sub3);
82      colorAt++;
83
84      triangles.push_back(sub1);
85      triangles.push_back(sub2);
86      triangles.push_back(sub3);
87
88      ftree(sub1, depth - 1);
89      ftree(sub2, depth - 1);
90      ftree(sub3, depth - 1);
```

```
91 }
```

Triangle.hpp

```
// Copyright 2022 <Justin Nguyen>
#ifndef _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS3_TRIANGLE_HPP_
#define _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS3_TRIANGLE_HPP_

#include <string>
#include <SFML/Graphics.hpp>
using sf::Vector2f;

class Triangle: public sf::Drawable {
 public:
    explicit Triangle(double = 1, Vector2f = Vector2f());
    double GetSideLength(unsigned int side) const;

    Vector2f GetPosition() const {return trigon.getPosition();}
    Vector2f GetPointPosition(size_t at) const {return trigon.getPoint(at);}

    void SetFillColor(sf::Color color) {trigon.setFillColor(color);}
    void SetOutlineColor(sf::Color color) {trigon.setOutlineColor(color);}
    void SetOutlineWidth(double width) {trigon.setOutlineThickness(width);}

 private:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;  //NOLINT
    sf::ConvexShape trigon;
    double a, b, c;
};

#endif  // _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS3_TRIANGLE_HPP_
```

Triangle.cpp

```
// Copyright 2022 <Justin Nguyen>
#include "Triangle.hpp"

#include <cmath>
#include <algorithm>
#include <iostream>
#include <string>

#include <SFML/Graphics.hpp>

const unsigned int NUM_TRI_SIDES(3);

using sf::Vector2f;

```

```
15  Triangle::Triangle(double sidelength, Vector2f pos)
16  :trigon(NUM_TRI_SIDES), c(sidelength) {
17      trigon.setPoint(0, pos);  //  left most
18      trigon.setPoint(1, pos + Vector2f(c, 0));  //  right most
19      trigon.setPoint(2, pos + Vector2f(c/2, c));  //  bottom most
20  }
21
22  double Triangle::GetSideLength(unsigned int side) const {
23      if (side == 0 || side > NUM_TRI_SIDES)
24          throw std::invalid_argument(
25              "Invalid triangle side. Side: " + std::to_string(side));
26      switch (side) {
27          case 1:
28              return a;
29          case 2:
30              return b;
31          case 3:
32              return c;
33      }
34      return c;
35  }
36
37  void Triangle::draw(sf::RenderTarget& target, sf::RenderStates states) const {
38      target.draw(trigon, states);
39  }
```

PS4: Circular Buffer + StringSound

Description:

In this project, I created a circular buffer, which is a fixed sized array that wraps back on itself. Using this data structure, I was then able to simulate a guitar string being plucked.

Discussion:

A guitar string is basically a bunch of vibrations. A soundwave is actually a bunch of random, discrete values.

This assignment simulates this with a circular buffer. By filling up a buffer with some random values bounded by [-2^15, 2^15-1], we can achieve a similar effect.

This assignment was my hardest one. My circular buffer did originally wrap around. So given an input of [1..10], it would make an array that went [6-10,1-5]. This was apparently the wrong design choice. The correct one was just to do it as [1..10]. What happened when I went to go do part B was it played an ear-screeching noise. Once I fixed the buffer, it played the correct sounds, an artificial but gentle guitar pluck.

This project was unit-tested. The ring buffer was required to pass some tests. It was tested to ensure its internals matched an expected output, and that it threw the correct exceptions.

StringSound was tested for: 1. StringSound(double) checks to see if its only argument is zero, because this constructor has to use its argument as a divisor. 2. StringSound(vector) checks to see if its vector is empty, because what is the point of having an empty vector. 3. StringSound::tic() will raise an exception since StringSound::pluck() must be called first.

Accomplished:

Making a circiular buffer and simulating a guitar string being plucked.

Learned:

- Concept of a circular buffer
- How sound works
- How to simulate a sound
- How to bind keys to an action

Key Algorithms/Data Structures/OO Designs:

- Model-View-Controller
  - The user, through some binded keys, controlled the model, and the model updated the view, or in this case, the audio for the user.

- Ring Buffer
  - A buffer that loops back on itself. On this project, it was used to hold sound data for playing the guitar string.

Uncompleted/Not Working Features:

Everything for this project was completed, and works.

Makefile

```
EXE_NAME = KSGuitarSim
OBJ = KSGuitarSim.o CircularBuffer.o StringSound.o
DEP = CircularBuffer.hpp StringSound.hpp
LIBS = -lsfml-system -lsfml-graphics -lsfml-window -lsfml-audio
TAR_NAME = PS4b_JustinNguyen.tar.gz
CC = g++
CPPV = c++17
FLAGS = -Wall -Werror -pedantic -std=$(CPPV)

LL = cpplint
LFLAGS = --filter=-runtime/references --filter=-build/$(CPPV)
#DEBUG = -g -ggdb3

all: $(EXE_NAME)

$(EXE_NAME): $(OBJ)
    $(CC) $(FLAGS) $(DEBUG) -o $@ $^ $(LIBS)

%.o: %.cpp %.hpp
    $(CC) $(FLAGS) $(DEBUG) -c $<


%.o: %.cpp
    $(CC) $(FLAGS) $(DEBUG) -c $^

run: # shortcut for you
    ./$(EXE_NAME)

lint:
    $(LL) $(LFLAGS) *.*pp

dist:
    tar --exclude='.vscode' -czvf ../$(TAR_NAME) .
    mv ../$(TAR_NAME) .
clean:
    rm -f $(OBJ) *.gch
distclean: clean
    rm -f $(EXE_NAME) $(TAR_NAME)
```

main.cpp

```
// Copyright 2022 Justin Nguyen
#include <cmath>
#include <map>
#include <vector>
#include <functional>
#include <memory>
#include <string>

#include <iostream>

#include "StringSound.hpp"

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

typedef sf::Keyboard::Key KEYCODE;

using std::vector;
using std::unique_ptr;
using std::make_unique;

const unsigned int CONCERT_A(220);
const int SAMPLING_RATE(44100);

const vector<KEYCODE> PIANO_KEYS({
    KEYCODE::Q, KEYCODE::Num2, KEYCODE::W,
    KEYCODE::E, KEYCODE::Num4, KEYCODE::R,
    KEYCODE::Num5, KEYCODE::T, KEYCODE::Y,
    KEYCODE::Num7, KEYCODE::U, KEYCODE::Num8,
    KEYCODE::I, KEYCODE::Num9, KEYCODE::O,
    KEYCODE::P, KEYCODE::Hyphen, KEYCODE::LBracket,
    KEYCODE::Equal, KEYCODE::Z, KEYCODE::X,
    KEYCODE::D, KEYCODE::C, KEYCODE::F,
    KEYCODE::V, KEYCODE::G, KEYCODE::B,
    KEYCODE::N, KEYCODE::J, KEYCODE::M,
    KEYCODE::K, KEYCODE::Comma, KEYCODE::Period,
    KEYCODE::Semicolon, KEYCODE::Slash, KEYCODE::Quote,
    KEYCODE::Space,
});

double getFrequency(double delta);
vector<sf::Int16> makeSamples(StringSound& gs); // NOLINT
```

```
45
46  const vector<std::string> NOTE_NAMES(
47      {"A", "A#/Bb", "B", "C", "C#/Db", "D", "D#/Eb", "E", "F", "F#/Gb", "G"});
48
49  int main(void) {
50      vector<unique_ptr<vector<sf::Int16>>> svvi16;
51      vector<unique_ptr<sf::SoundBuffer>> sbs;
52      vector<unique_ptr<sf::Sound>> snds;
53
54      sf::RenderWindow window(sf::VideoMode(300, 300), "KSGuitarSim");
55
56      std::map<KEYCODE,
57          std::pair<size_t, std::function<void(sf::Sound&)>>> pianoKeys;
58
59
60      size_t i = 0;
61      for (auto key : PIANO_KEYS) {
62          StringSound ssi(getFrequency(i));
63          svvi16.push_back(make_unique<vector<sf::Int16>>(makeSamples(ssi)));
64
65          sbs.push_back(make_unique<sf::SoundBuffer>());
66          sbs.back()->loadFromSamples
67              (svvi16.back()->data(), svvi16.back()->size(), 2, SAMPLING_RATE);
68
69          snds.push_back(make_unique<sf::Sound>(sf::Sound(*sbs.back())));
70
71          pianoKeys[key] =
72              std::make_pair(i++, [](sf::Sound& s){s.play();});
73      }
74
75      while (window.isOpen()) {
76          sf::Event e;
77          while (window.pollEvent(e)) {
78              switch (e.type) {
79              case sf::Event::Closed:
80                  window.close();
81                  break;
82              case sf::Event::KeyPressed: {
83                  auto keySound = pianoKeys[e.key.code];
84                  std::cout <<
85                      NOTE_NAMES.at(keySound.first % NOTE_NAMES.size()) << " ";
86                  try {
87                      keySound.second(*snds.at(keySound.first));
88                  } catch(const std::bad_function_call&) {
89                      std::cout << "Invalid key." << std::endl;
90                  }
```

```
91              }
92              default:
93                  break;
94              }
95          }
96          window.clear();
97          window.display();
98      }
99
100     return 0;
101 }
102
103 double getFrequency(double delta) {
104     return CONCERT_A*2.f * pow(2.f, (delta - 24.f)/12.f);
105 }
106
107 vector<sf::Int16> makeSamples(StringSound& gs) { //  NOLINT
108     vector<sf::Int16> samples;
109     gs.pluck();
110     int duration(8);  // seconds
111     for (int i = 0; i < SAMPLING_RATE * duration; i++) {
112         gs.tic();
113         samples.push_back(gs.sample());
114     }
115     return samples;
116 }
```

StringSound.hpp

```
// Copyright 2022 Justin Nguyen
#ifndef _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS4B_STRINGSOUND_HPP_
#define _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS4B_STRINGSOUND_HPP_

#include <stdint.h>

#include <vector>
#include <memory>

#include "CircularBuffer.hpp"

namespace SSC {  // string sound constants
    const unsigned int SAMPLING_RATE(44100);
    const double DECAY_FACTOR(0.996);
}

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

    // create a guitar string with
    // size and initial values are given by
    // the vector
    explicit StringSound(const std::vector<int16_t>& init);

    StringSound (const StringSound& obj) = delete;  // no copy const
    ~StringSound() {}

    // pluck the guitar string by replacing
    // the buffer with random values,
    // representing white noise
    void pluck();

    void tic();  // advance the simulation one time step

    // return the current sample
    int16_t sample() const {return _cb->peek();}

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

    friend std::ostream& operator<<(std::ostream& out,
```

```
45      const StringSound& ss) {
46          return out << *(ss._cb);
47      }
48
49   private:
50      std::unique_ptr<CircularBuffer> _cb;
51      int _time;
52  };
53
54  #endif  // _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS4B_STRINGSOUND_HPP_
```

StringSound.cpp

```
1   // Copyright 2022 Justin Nguyen
2
3   #include <cmath>
4   #include <vector>
5   #include <random>
6   #include <chrono> // NOLINT
7   #include <limits>
8   #include <stdexcept>
9
10  #include "StringSound.hpp"
11  #include "CircularBuffer.hpp"
12
13  StringSound::StringSound(double frequency):
14  _cb(new CircularBuffer(ceil(SSC::SAMPLING_RATE/frequency))), _time(0) {
15      if (frequency == 0)
16          throw std::invalid_argument(
17          "StringSound(double): frequency cannot be zero.");
18  }
19
20  StringSound::StringSound(const std::vector<int16_t>& init):
21  _cb(new CircularBuffer(init.size())), _time(0) {
22      if (init.size() == 0)
23          throw std::invalid_argument(
24          "StringSound(vector): vector cannot be empty.");
25
26      for (int16_t n : init)
27          _cb->enqueue(n);
28  }
29
30  void StringSound::pluck() {
31      std::mt19937 rng(
32          std::chrono::system_clock::now().time_since_epoch().count());
33
```

```
34      std::uniform_int_distribution<int16_t>
35      dist(std::numeric_limits<int16_t>::min(),
36      std::numeric_limits<int16_t>::max());
37
38      _cb->empty();
39
40      for (size_t i = 0; i < _cb->capacity(); i++)
41          _cb->enqueue(dist(rng));
42  }
43
44  void StringSound::tic() {
45      if (_cb->isEmpty())
46          throw std::runtime_error(
47          "StringSound::tic(): cannot tic an empty ring buffer.");
48
49      _cb->enqueue(SSC::DECAY_FACTOR*0.5*((_cb->dequeue() + _cb->peek())));
50      _time++;
51  }
```

CircularBuffer.hpp

```cpp
#ifndef _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS4B_CIRCULARBUFFER_HPP_
#define _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS4B_CIRCULARBUFFER_HPP_

// Copyright 2022 Justin Nguyen
#include <stdlib.h>
#include <stdint.h>

#include <stdexcept>
#include <deque>
#include <iostream>

class CircularBuffer {
 public:
    explicit CircularBuffer(int capacity)
    : s(0), c(capacity), deque(capacity, 0) {
        if (capacity < 1)
            throw std::invalid_argument(
                "CircularBuffer(int): capacity must be greater than zero.");
    }
    size_t size() const {return s;}
    size_t capacity() const {return c;}

    bool isEmpty() const {return s == 0;}
    bool isFull() const {return s == c;}

    void enqueue(int16_t x);
    int16_t dequeue();
    int16_t peek() const;

    void empty();

    friend std::ostream& operator<<(std::ostream& out,
    const CircularBuffer& c) {
        for (size_t i = 0; i < c.s; i++)
            out << c.deque.at(i) << ", ";
        return out;
    }

 private:
    size_t s;
    const size_t c;
    std::deque<int16_t> deque;
};

```

```
45 #endif  // _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS4B_CIRCULARBUFFER_HPP_
```

CircularBuffer.cpp

```
1  // Copyright 2022 Justin Nguyen
2  #include <stdlib.h>
3  #include <stdint.h>
4  #include <iostream>
5
6  #include "CircularBuffer.hpp"
7
8  void CircularBuffer::enqueue(int16_t x) {
9      if (isFull())
10         throw std::runtime_error("enqueue: can't enqueue to a full ring");
11     deque.push_back(x);
12     s++;
13 }
14
15 int16_t CircularBuffer::dequeue() {
16     if (isEmpty())
17         throw std::runtime_error("dequeue: can't dequeue to an empty ring");
18     int16_t ret = peek();
19     deque.pop_front();
20     s--;
21     return ret;
22 }
23
24 int16_t CircularBuffer::peek() const {
25     if (isEmpty())
26         throw std::runtime_error("peek: can't peek an empty ring");
27     return deque.front();
28 }
29
30 void CircularBuffer::empty() {
31     s = 0;
32 }
```

UnitTest.cpp

```
// Copyright 2022 Justin Nguyen
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main

#include <stdint.h>

#include <iostream>
#include <vector>

#include "StringSound.hpp"
#include <boost/test/unit_test.hpp>
#include <boost/test/tools/output_test_stream.hpp>

BOOST_AUTO_TEST_CASE(StringSoundTest) {
    const size_t SIZE(10);
    CircularBuffer a(SIZE);

    StringSound ss1(44100.f/220.f);
    BOOST_REQUIRE_THROW(StringSound(0), std::invalid_argument);

    BOOST_REQUIRE_THROW(StringSound(std::vector<int16_t>()),
    std::invalid_argument);

    BOOST_REQUIRE_THROW(ss1.tic(), std::runtime_error);
    ss1.pluck();
    BOOST_REQUIRE_NO_THROW(ss1.tic());



    // bad constructor test
    BOOST_REQUIRE_THROW(CircularBuffer(0), std::invalid_argument);

    BOOST_REQUIRE(a.isEmpty());  // should be empty on start
    BOOST_REQUIRE(!a.isFull());  // should not be full
    BOOST_REQUIRE(a.capacity() == SIZE);
    BOOST_REQUIRE(a.size() == 0);

    // throw on dequeuing empty buffer
    BOOST_REQUIRE_THROW(a.dequeue(), std::runtime_error);

    // throw on peeking on empty buffer
    BOOST_REQUIRE_THROW(a.peek(), std::runtime_error);

    BOOST_REQUIRE_NO_THROW(a.enqueue(0));
```

```
45
46      BOOST_REQUIRE_NO_THROW(a.peek());
47      BOOST_REQUIRE_NO_THROW(a.dequeue());
48
49      // should not be throwing here. we are within bounds
50      BOOST_REQUIRE_NO_THROW(
51      for (size_t i = 0; i < SIZE; i++)
52          a.enqueue(i));
53
54      BOOST_REQUIRE(a.isFull());
55
56      // throw on queuing full buffer
57      BOOST_REQUIRE_THROW(a.enqueue(SIZE + 1), std::runtime_error);
58
59      BOOST_REQUIRE_NO_THROW(
60      for (size_t i = 0; i < SIZE; i++)
61          a.dequeue());
62  }
```

PS5: DNA Alignment

Description:

In this project, a program was built to check the alignment of two DNA sequences. DNA sequences are made up of Adenine, Thymine, Guanine and Cytosine macromolecules. The objective was to see how well two DNA sequences matched, and to also align them.

How alignment was scored was based on such rubric:

- Match: 0
- Mismatch: 1
- Gap Insertion: 2

Where a score closest to 0 was considered to be the most optimal alignment.

This assignment was also benchmarked to check for performance.

Discussion:

I implemented the Needleman-Wunsch Algorithm for this project, with space and time complexity of O(m*n), which is generally O(nˆ2). The Needleman-Wunsch Algorithm exhausts all matches in a neat m **x** n matrix. It is incredibly efficient at this task, and is not too difficult to implement.

My interpretation of this algorithm:

Instead of finding out an almost infinite number of possibilities, the matrix solves all of this and gives NxM possibilities. The matrix then puts the progressive cost of such a combination in its cells. Alignment is then found by backtracking, and trying to sum up the previous cells with the associated cost to the current cell by reversing the algorithm.

There were other algorithms, such as the recursive approach, though this would've taken too much time (O(2ˆn)). Hirschberg, which is a modification of Needleman-Wunsch, uses less space (approx. O(n)). I unfortunately did not get around to doing this.

Benchmarks:

Input Size	V0 Runtime	V1 Runtime	V2 Runtime
2500	1.14763s	0.485189s	0.397811s
5000	4.65864s	1.93819s	1.48295s
10000	18.0051s	7.70221s	5.87246s
20000	73.1131s	31.5668s	23.7488s
50000	788.067s	332.243	251.403s

- (V0) Initially, my original program actually copied references to the first row and column elements from the matrix. This made my program run

extremely slowly. Some functions were also refactored for performance.

- (V1) After fixing all of these, the running times and memory usage of my program greatly improved. This was a great lesson for me to avoid copying, which is an expensive operation.
- (V2) Also, as a last measure, I stopped using vector::at() in the internal representation of my Matrix class. Surprisingly, this did improve running times by no insignificant amount. Bounds checking, in tight loops, is also expensive.

After optimizing my program, I performed some time and memory complexity analysis using some special tools. I first analyzed it from a mathematical perspective. The amount of memory used by the program lines up with the memory complexity of O(m*n). Time complexity has a similar complexity due to 3mn comparisions.

After some mathematical analysis, I performed some real analysis using some profiling tools. Memory usage was scanned, and matched up with the expected results very closely. Time complexity almost exactly matched expected complexity. The actual graph aligned closely with an O(n^2) graph.

On the side, I did some profiling using kcachegrind and found out my Matrix::at() method was consuming a decent amount of time.

Accomplished:

Aligning two strings, and gaining a metric to gauge how similar they are, and profiling a program.

Learned:

- How to align two strings
- How to check how similar two strings are
- How to benchmark
- How to measure the running time of a program
- How to check the memory usage of a program
- How to profile a program

Key Algorithms/Data Structures/OO Designs:

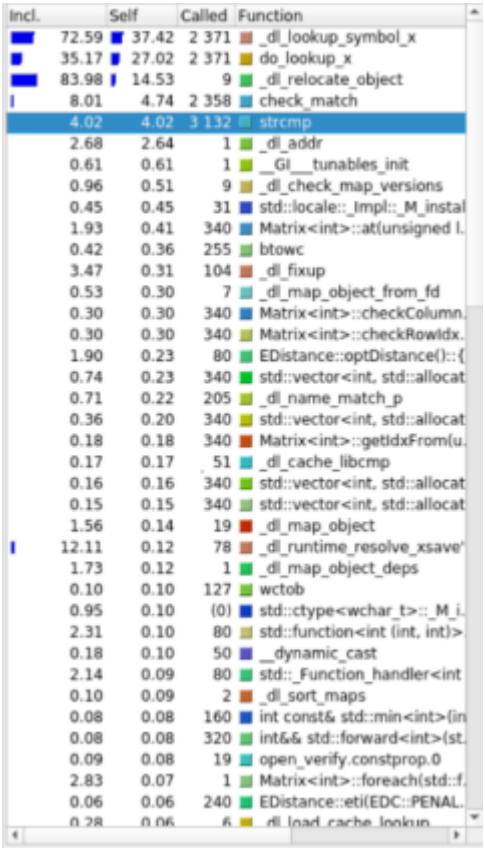
- Needleman-Wunsch Algorithm
  - An efficient algorithm that uses an O(n **x** m) matrix to calculate the alignment of two strings.
- Matrix structure
  - For this project, I made my own templated matrix. A matrix, in a mathematical sense, is a system of linear equations.

Uncompleted/Not Working Features:

Everything for this project was completed, and works.

Output: (evidence the code ran)

Edit distance = 2
X Y cost
=====
A T 1
A A 0
C _ 2
A A 0
G G 0
T G 1
T T 0
A _ 2
C C 0
C A 1



Incl.	Self	Called	Function
72.59	37.42	2 371	_dl_lookup_symbol_x
35.17	27.02	2 371	do_lookup_x
83.98	14.53	9	_dl_relocate_object
8.01	4.74	2 358	check_match
4.02	4.02	3 132	strcmp
2.68	2.64	1	_dl_addr
0.61	0.61	1	__GI___tunables_init
0.96	0.51	9	_dl_check_map_versions
0.45	0.45	31	std::locale::_Impl::_M_instal
1.93	0.41	340	Matrix<int>::at(unsigned l.
0.42	0.36	255	btowc
3.47	0.31	104	_dl_fixup
0.53	0.30	7	_dl_map_object_from_fd
0.30	0.30	340	Matrix<int>::checkColumn.
0.30	0.30	340	Matrix<int>::checkRowIdx.
1.90	0.23	80	EDistance::optDistance()::{
0.74	0.23	340	std::vector<int, std::allocat
0.71	0.22	205	_dl_name_match_p
0.36	0.20	340	std::vector<int, std::allocat
0.18	0.18	340	Matrix<int>::getIdxFrom(u.
0.17	0.17	51	_dl_cache_libcmp
0.16	0.16	340	std::vector<int, std::allocat
0.15	0.15	340	std::vector<int, std::allocat
1.56	0.14	19	_dl_map_object
12.11	0.12	78	_dl_runtime_resolve_xsave'
1.73	0.12	1	_dl_map_object_deps
0.10	0.10	127	wctob
0.95	0.10	(0)	std::ctype<wchar_t>::_M_i.
2.31	0.10	80	std::function<int (int, int)>
0.18	0.10	50	__dynamic_cast
2.14	0.09	80	std::_Function_handler<int
0.10	0.09	2	_dl_sort_maps
0.08	0.08	160	int const& std::min<int>(in
0.08	0.08	320	int&& std::forward<int>(st.
0.09	0.08	19	open_verify.constprop.0
2.83	0.07	1	Matrix<int>::foreach(std::f.
0.06	0.06	240	EDistance::eti(EDC::PENAL.
0.28	0.06	6	_dl_load_cache_lookup

Figure 6: *KCacheGrind results.*

Makefile

```
EXE_NAME = EDistance
OBJ = main.o EDistance.o
LIBS = -lsfml-system #-lsfml-graphics -lsfml-window #-lsfml-audio
TAR_NAME = PS5_JustinNguyen.tar.gz
DEP = Matrix.hpp
CC = g++
CPPV = c++17
#DEBUG = -g -ggdb3 -fmax-errors=1
FLAGS = -Wall -Werror -pedantic -std=$(CPPV) $(DEBUG) -g

LL = cpplint
LFLAGS = --filter=-runtime/references --filter=-build/$(CPPV)

VFLAGS = --leakcheck=full

all: $(EXE_NAME)

$(EXE_NAME): $(OBJ)
    $(CC) $(FLAGS) -o $@ $^ $(LIBS)

%.o: %.cpp %.hpp $(DEP)
    $(CC) $(FLAGS) -c $<
%.o: %.cpp $(DEP)
    $(CC) $(FLAGS) -c $<

run: # shortcut for you
    ./$(EXE_NAME) < studentInput.txt > output.txt

lint:
    $(LL) $(LFLAGS) *.*pp
valgrind:
    valgrind ./$(EXE_NAME) < studentInput.txt

dist:
    tar --exclude='.vscode' -czvf ../$(TAR_NAME) .
    mv ../$(TAR_NAME) .
clean:
    rm -f $(OBJ) callgrind*
distclean: clean
    rm -f $(EXE_NAME) $(TAR_NAME)
```

main.cpp

```cpp
// Copyright 2022 Justin Nguyen
#include <iostream>
#include <string>

#include "Matrix.hpp"
#include "EDistance.hpp"

#include <SFML/System/Clock.hpp>

int main(void) {
// sf::Clock clock;

string a, b;
std::cin >> a >> b;

// std::cout << "Input size ((strA + strB).size()): "
// << a.size() + b.size()
// << " chars" << std::endl;

// std::cout << "Processing..." << std::endl;

EDistance ed(a, b);

std::cout << "Edit distance = " << ed.optDistance();

// std::cout << "Elapsed time: "
// << clock.getElapsedTime().asSeconds() << 's' << std::endl;

std::cout << ed.alignment() << std::endl;

return 0;
}
```

EDistance.hpp

```
// Copyright 2022 Justin Nguyen
#ifndef _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS5_EDISTANCE_HPP_
#define _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS5_EDISTANCE_HPP_

#include <string>
#include <algorithm>
#include <stdexcept>

#include "Matrix.hpp"

using std::string;

namespace EDC {
    enum class PENALTIES {MATCH, MISMATCH, GAP};
}

using EDC::PENALTIES;

class EDistance {
 public:
    EDistance(const string& strA, const string& strB):
    a(string(" ") + max(strA, strB)), b(string(" ") + min(strA, strB)),
    A(a.size(), b.size(), 0) {
        if (strA.size() == 0 || strB.size() == 0)
            throw std::runtime_error(
            "EDistance(a,b): Both strings must have at least one character.");
    }

    int optDistance();
    string alignment();

 private:
    const string a;
    const string b;

    Matrix<int> A;
    static int eti(PENALTIES p) {
        return static_cast<int>(p);
    }
    static int penalty(char a, char b) {
        return a == b ?
        eti(PENALTIES::MATCH) :
        eti(PENALTIES::MISMATCH);
    }
```

```
45      static int min(int a, int b, int c) {
46          return std::min(a, std::min(b, c));
47      }
48      static string min(const string& strA, const string& strB) {
49          if (strA.size() == strB.size())
50              return strB;
51          return strA.size() < strB.size() ? strA : strB;
52      }
53      static string max(const string& strA, const string& strB) {
54          if (strA.size() == strB.size())
55              return strA;
56          return strA.size() > strB.size() ? strA : strB;
57      }
58  };
59
60  #endif  // _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS5_EDISTANCE_HPP_
```

EDistance.cpp

```
1   // Copyright 2022 Justin Nguyen
2   #include <string>
3   #include <sstream>
4   #include <vector>
5   #include <algorithm>
6   #include <functional>
7   #include <utility>
8
9   #include "EDistance.hpp"
10
11  using EDC::PENALTIES;
12  using std::make_pair;
13
14  namespace FMAT {
15      const char SPACE(' ');
16      const char GAP('-');
17      const unsigned int WIDTH(1);
18  }
19
20  int f2x(int x) {
21      return x*static_cast<int>(PENALTIES::GAP);
22  }
23
24  void fill(std::vector<std::vector<int>::iterator>&v, std::function<int(int)> f) {  // NOLIN
25      for (size_t i = 0; i < v.size(); i++)
26          *v[i] = f(i);
27  }
```

```
28
29  int EDistance::optDistance() {  // the original NW algo
30      A.forRow(0, f2x);
31      A.forColumn(0, f2x);
32
33      A.foreach([&](size_t i, size_t j) {
34          return min(
35              A.at(i - 1, j) + eti(PENALTIES::GAP),  // up
36              A.at(i - 1, j - 1) + penalty(a.at(i), b.at(j)),  // diagonal
37              A.at(i, j - 1) + eti(PENALTIES::GAP));  // left
38      }, 1, 1);
39
40      return *A.rbegin();  // returning last item at A[M][N]
41  }
42
43  string EDistance::alignment() {
44      std::stringstream alignTbl;
45      int i = A.dimensions().first - 1, j = A.dimensions().second - 1;
46      string aligned;
47      while (i != 0 && j != 0) {
48          int Aij(A.at(i, j));
49          int cost(eti(PENALTIES::MATCH));
50          char x(0), y(0);
51
52          if (Aij == A.at(i-1, j) + eti(PENALTIES::GAP)) {  // up
53              cost = eti(PENALTIES::GAP);
54              x = a.at(i--);
55              y = FMAT::GAP;
56          } else if (Aij == A.at(i, j-1) + eti(PENALTIES::GAP)) {  // left
57              cost = eti(PENALTIES::GAP);
58              x = FMAT::GAP;
59              y = b.at(j--);
60          } else {
61              if ((Aij == A.at(i-1, j-1) + eti(PENALTIES::MISMATCH)))  // diagonal
62                  cost = eti(PENALTIES::MISMATCH);
63              x = a.at(i--);
64              y = b.at(j--);
65          }
66          alignTbl
67          << std::to_string(cost)
68          << string(FMAT::WIDTH, FMAT::SPACE)
69          << y
70          << string(FMAT::WIDTH, FMAT::SPACE)
71          << x
72          << '\n';
73      }
```

```
74      aligned = alignTbl.str();
75      std::reverse(aligned.begin(), aligned.end());
76      return aligned;
77  }
```

Matrix.hpp

```cpp
// Copyright 2022 Justin Nguyen
#ifndef _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS5_MATRIX_HPP_
#define _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS5_MATRIX_HPP_

#include <stdexcept>
#include <string>
#include <vector>
#include <iostream>
#include <iomanip>
#include <utility>
#include <functional>

namespace MC {
    const std::string MatrixDelimLeft("|");
    const std::string MatrixDelimRight("|");
    const std::string MatrixSeparator(" ");
}

using std::vector;

template<typename T>
class Matrix {
 public:
    Matrix(size_t rows, size_t columns, T filler = 0):
    m_(rows), n_(columns), A(rows*columns, filler) {}

    std::pair<size_t, size_t> dimensions() const {
        return std::make_pair(m_, n_);
    }

    size_t size() const {return m_*n_;}

    T& at(size_t i, size_t j) {
        checkRowIdx_(i);
        checkColumnIdx_(j);
        return A[getIdxFrom(i, j)];
    }
    T& operator()(size_t i, size_t j) {
        return A[getIdxFrom(i, j)];
    }
    T& operator[](std::pair<size_t, size_t> ij) {  // no bounds check, caution!
        return operator()(ij.first, ij.second);
    }

```

```
45      void forRow(size_t i, std::function<T(size_t)> f) {
46          for (size_t j = 0; j < n_; j++)
47              at(i, j) = f(j);
48      }
49
50      void forColumn(size_t j, std::function<T(size_t)> f) {
51          for (size_t i = 0; i < m_; i++)
52              at(i, j) = f(i);
53      }
54
55      void foreach(std::function<T(int, int)> f,
56      size_t rstart = 0, size_t cstart = 0, size_t rend = -1, size_t cend = -1) {
57          if (rend == size_t(-1))
58              rend = m_;
59          if (cend == size_t(-1))
60              cend = n_;
61
62          for (size_t i = cstart; i < rend; i++)
63              for (size_t j = rstart; j < cend; j++)
64                  at(i, j) = f(i, j);
65      }
66
67      void rforeach(std::function<T(int, int)> f,
68      int rstart = -1, int cstart = -1, int rend = 0, int cend = 0) {
69          if (rstart == -1)
70              rstart = m_ - 1;
71          if (cstart == -1)
72              cstart = n_ - 1;
73
74          for (int i = rstart; i >= rend; i--)
75              for (int j = cstart; j >= cend; j--)
76                  at(i, j) = f(i, j);
77      }
78
79      auto begin() const {return A.begin();}
80      auto end() const {return A.end();}
81
82      auto rbegin() const {return A.rbegin();}
83      auto rend() const {return A.rend();}
84
85      friend std::ostream& operator<<(std::ostream& out, const Matrix<T>& A) {
86          size_t j = 0;
87
88          for (const T& val : A) {
89              out << std::setw(A.n_/3)
90                  << std::setfill(MC::MatrixSeparator.at(0));
```

```
91
92              if (j == 0)
93                  out << MC::MatrixDelimLeft;
94
95              out << val;
96
97              if (j + 1 == A.dimensions().second)  // EOL
98                  out << MC::MatrixDelimRight << std::endl;
99
100             ++j %= A.dimensions().second;
101         }
102
103         return out;
104     }
105
106  private:
107     const size_t m_, n_;
108     vector<T> A;
109     size_t getIdxFrom(size_t i, size_t j) {
110         return i * n_ + j;
111     }
112     void checkRowIdx_(size_t i) {
113         if (i >= m_)
114             throw std::runtime_error(
115                 "Matrix: Row index cannot exceed row length, which is "
116                 + std::to_string(m_-1));
117     }
118     void checkColumnIdx_(size_t j) {
119         if (j >= n_)
120             throw std::runtime_error(
121                 "Matrix: Column index cannot exceed column length, which is "
122                 + std::to_string(n_-1));
123     }
124 };
125
126 #endif  // _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS5_MATRIX_HPP_
```

PS6: Random Writer

Description:

In this project, I created a probabilistic text generator using Markov chains. This text generator would take three parameters: the source text, the order k of the model, and a kgram.

In essence, a Markov chain is probabilistic mathematical system that heavily depends on its previous state to reach its next state. An example could be a shop that sells pizza, hot dogs and hamburgers on certain days. We could know when the shop sells hotdogs today if the last day they sold hamburgers, because the shop is more likely to sell hotdogs if they sold hamburgers yesterday.

RandWriter follows a similar principle, where given a kgram (a substring of the source text), it selected a letter that followed the kgram given its frequency in the text. For example, given the kgram "the", possible letters that could follow it are 'n', 'r', 'o'.

Discussion:

The first thing I did when starting my project was designing a function to find all substrings of a given substring in the source text. My entire project is actually designed around this function:

```
std::vector<string::const_iterator>
RandWriter::getAllKgramPos(const string& kgram) const {
    checkKgram(kgram);
    if (kgramMap.count(kgram) == 1)
        return kgramMap.at(kgram);

    std::vector<string::const_iterator> substrPos;
    auto idx = t.begin();
    while ((idx = std::search(idx, t.end(), kgram.begin(), kgram.end()))
    != t.end())
        substrPos.push_back(idx += kgram.size());
    return substrPos;
}
```

I do not know its complexity, though it is probably around in the ballpark of O(n^2). This function searches for all instances of a substring in my source text, and returns a vector of their positions. This made creating the other required methods much easier. `RandWriter::freq(string)` only needs to call this function, and get the size of vector generated by it. The true guts of this project comes from `RandWriter::generate(string)`

```
string RandWriter::generate(const string& kgram, int L) {
    checkKgram(kgram);
    string kngram(kgram);
```

```
    string ret(kgram);

    while (ret.size() < size_t(L)) {
        ret.push_back(kRand(kngram));
        kngram.push_back(ret.back());
        kngram.erase(kngram.begin());
    }

    return ret;
}
```

By adding a new character, and removing the old one, this resembles a Markov chain, where the current state depends on the previous state.

When the text is generated, interesting results come at different orders. Here are orders 0-5.

Order	Generated Text
0	swdlmmmTctepo os ol;uo'uk gi uh ayrmx d" aha esty gmegeown tta s,s
1	To best The-hawirdin whasisitr, ar Sincad ked me wed pas f, bldemetang He Ohtyatcay fo ed stotan ss
2	A dild of thelleces sin toody, brife; yousput thad hown dise so fore vast. But sty hou
3	THE finds ripping which or aroup overy shelp mur she she stofferificing here flockets said:
4	THE sun dig unreason were or sever Two."
5	THAT night to be profound!" Tom could go and-a twig almost

At order 0, RandWriter will just pick letters based on their frequency in the text. In my implementation, it was simple as picking a random letter from the source text.

At order 1-3, it is still nonsense but we can begin to see words being formed. It almost looks like english. At order 4-5, the generated text, while nonsensical, begins to resemble its source.

Accomplished:

Creating a random text generator.

Learned:

- How Markov chains work
- How to generate random text

Key Algorithms/Data Structures/OO Designs:

- std::set<>
  - While not the most key component, this structure was very useful in keeping track of what letters were used in the text. As one knows, a set is a container of unique elements, so it was perfect for this task.
- std::map
  - A very key data structure used to record the frequency of a k+1gram. My structure was set up like: kgram: [“the”] = { [‘n’] = 5, [‘r’] = 4 }

Uncompleted/Not Working Features:

Everything for this project was completed, and works.

Output: (evidence the code ran)

- Inputs:
  - Source Text: The Adventures of Tom Sawyer by Mark Twain
  - Order: 6
  - Length: 200 characters
- Output:
  - Most of grapevine, and then learn you drove the handled brush another with a moan. Aunt Polly raised forward; and by." "Oh, never separation, and shabby village, when he began to pity for we are only a smile, but night, to...

Makefile

```
EXE_NAME = TextWriter
OBJ = main.o RandWriter.o
LIBS = -lsfml-system -lboost_unit_test_framework #-lsfml-graphics -lsfml-window #-lsfml-audi
TAR_NAME = PS6_JustinNguyen.tar.gz
DEP = RandWriter.hpp
CC = g++
CPPV = c++17
#DEBUG = -g -ggdb3 -fmax-errors=1
FLAGS = -Wall -Werror -pedantic -std=$(CPPV) $(DEBUG)

LL = cpplint
LFLAGS = --filter=-runtime/references --filter=-build/$(CPPV)

VFLAGS = --leakcheck=full

all: $(EXE_NAME)

$(EXE_NAME): $(OBJ)
    $(CC) $(FLAGS) -o $@ $^ $(LIBS)

%.o: %.cpp %.hpp $(DEP)
    $(CC) $(FLAGS) -c $<
%.o: %.cpp $(DEP)
    $(CC) $(FLAGS) -c $<

run: # shortcut for you
    ./$(EXE_NAME) 6 200 < texts/tomsawyer.txt > output.txt

lint:
    $(LL) $(LFLAGS) *.*pp
valgrind:
    valgrind $(VFLAGS) ./$(EXE_NAME)

dist:
    tar --exclude='.vscode' -czvf ../$(TAR_NAME) .
    mv ../$(TAR_NAME) .
clean:
    rm -f $(OBJ)
distclean: clean
    rm -f $(EXE_NAME) $(TAR_NAME)
```

main.cpp

```cpp
// Copyright 2022 Justin Nguyen
#include <iostream>
#include <sstream>
#include <random>
#include <string>

#include "RandWriter.hpp"

int main(int argc, char** argv) {
    int order(std::stoi(argv[1])), length(std::stoi(argv[2]));

    std::stringstream ss;
    ss << std::cin.rdbuf();

    RandWriter rw(ss.str(), order);
    string kgram(ss.str());
    kgram = string(ss.str());
    kgram = string(kgram.begin(), kgram.begin() + order);

    std::cout << "GENERATED: " << rw.generate(kgram, length) << std::endl;

    std::cout << rw << std::endl;
    return 0;
}
```

RandWriter.hpp

```
// Copyright 2022 Justin Nguyen
#ifndef _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS6_RANDWRITER_HPP_
#define _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS6_RANDWRITER_HPP_

#include <iostream>
#include <chrono>  // NOLINT
#include <algorithm>
#include <random>
#include <iterator>

#include <vector>
#include <string>
#include <set>
#include <map>

using std::string;
using std::chrono::system_clock;

class RandWriter {
 public:
    RandWriter(const string& text, int k):
    t(text), order(k), rng(system_clock::now().time_since_epoch().count()) {
        using std::to_string;
        if (text.size() < size_t(order))
            throw std::invalid_argument(
            "Text size (which is " + to_string(text.size())
            + ") must be of length: " + to_string(k));
    }

    int orderK() const {return order;}

    int freq(const string& kgram) const;

    int freq(const string& kgram, char c) const;

    char kRand(const string& kgram);

    string generate(const string& kgram, int L);

    friend std::ostream& operator<<(std::ostream& out, const RandWriter& rw) {
        using std::endl;
        out << "Order: " << rw.order << endl;
        out << "Alphabet: ";
        std::copy(
```

```
45              rw.alphabet.begin(), rw.alphabet.end(),
46              std::ostream_iterator<char>(out, ""));
47          out << endl;
48          out << string(10, '-') << endl;
49
50          for (const auto& kgramPair : rw.letterFrequency) {
51              out << "K-GRAM: ";
52              out << kgramPair.first << endl;
53
54              out << "\tK+1-GRAM:" << endl;
55              for (const auto& letterCount : kgramPair.second)
56                  out << string(2, '\t')
57                  << letterCount.first
58                  << ": "
59                  << letterCount.second << endl;
60
61              out << string(10, '-') << endl;
62          }
63          return out;
64      }
65
66  private:
67      // removes all non-alphabetic characters
68      string sanitize(const string& str) const {
69          string copy(str);
70          copy.erase(
71              std::remove_if(copy.begin(), copy.end(),
72              [](unsigned char c) {
73                  return !std::isalpha(c);
74              }), copy.end());
75          return copy;
76      }
77      void checkKgram(const string& kgram) const {
78          if (kgram.size() != size_t(order))
79              throw std::runtime_error(
80              "Kgram \""
81              + sanitize(kgram)
82              + "\" must be length: " + std::to_string(order));
83      }
84      std::vector<string::const_iterator>
85      getAllKgramPos(const string& kgram) const;
86      const string t;
87      const int order;
88      std::map<string, std::vector<string::const_iterator>> kgramMap;
89      std::map<string, std::map<char, int>> letterFrequency;
90      std::set<char> alphabet;
```

```
    std::minstd_rand rng;
};

#endif  // _HOME_CAOTIN_SCHOOL_COMP4_PROJECTS_PS6_RANDWRITER_HPP_
```

RandWriter.cpp

```
// Copyright 2022 Justin Nguyen
#include <string>
#include <vector>
#include <map>

#include <iostream>
#include <utility>
#include <algorithm>
#include <random>
#include <stdexcept>

#include "RandWriter.hpp"

using std::string;

std::vector<string::const_iterator>
RandWriter::getAllKgramPos(const string& kgram) const {
    checkKgram(kgram);
    if (kgramMap.count(kgram) == 1)
        return kgramMap.at(kgram);

    std::vector<string::const_iterator> substrPos;
    auto idx = t.begin();
    while ((idx = std::search(idx, t.end(), kgram.begin(), kgram.end()))
    != t.end())
        substrPos.push_back(idx += kgram.size());
    return substrPos;
}

int RandWriter::freq(const string& kgram) const {
    return getAllKgramPos(kgram).size();
}

int RandWriter::freq(const string& kgram, char c) const {
    auto kgramPosList = getAllKgramPos(kgram);

    return std::count_if(kgramPosList.begin(), kgramPosList.end(),
    [&](auto kgramPosItr){
        return *kgramPosItr == c;
```

```
    });
}

char RandWriter::kRand(const string& kgram) {
    checkKgram(kgram);
    char ret;
    if (kgram.empty()) {
        ret = t.at(rng() % t.size());
    } else {
        std::vector<string::const_iterator>
        kgramPosList = getAllKgramPos(kgram);

        if (kgramMap.count(kgram) == 0)
            kgramMap[kgram] = kgramPosList;

        if (kgramPosList.empty())
            throw std::runtime_error(
                "Kgram: \"" + sanitize(kgram) + "\" does not exist.");

        ret = *kgramPosList.at(rng() % kgramPosList.size());
    }
    letterFrequency[kgram][ret]++;
    alphabet.insert(ret);
    alphabet.insert(kgram.begin(), kgram.end());
    return ret;
}

string RandWriter::generate(const string& kgram, int L) {
    checkKgram(kgram);
    string kngram(kgram);
    string ret(kgram);

    while (ret.size() < size_t(L)) {
        ret.push_back(kRand(kngram));
        kngram.push_back(ret.back());
        kngram.erase(kngram.begin());
    }

    return ret;
}
```

PS7: Kronos Time Parsing

Description:

In this project, a massive text file was parsed for two key strings. A Kronos Time log has two indicators of a successful boot up. The first being "(log.c.166) server started", and the second being "oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080." These must be matched, with no nesting, in order for a valid boot up.

Discussion:

For this project, as the text file was so massive, and the two key strings were not nested, I chose to read the file line by line. This approach simplified many steps. I did not have to load the entire file into a string, or try to manage its input size. For the regexes, I just had to match "log.c.166" and "oejs.AbstractConnector". I verified this by making sure there was no other string in the file that looked similar to these two. The regexes I came up with were:

- `\(log.c.166\) server started\s?$` - (log.c.166) server started
- `oejs\.AbstractConnector:Started` - oejs.AbstractConnector:Started

The actual implementation was not too bad. First, I had to look for "log.c.166". Once this was found, a boolean flag was set to indicate it has been found. In the case of a failure, the completion message will not be there. This boolean flag is set in the case where we find another log.c.166, instead of the correct end string. The program will branch here and input the first log.c.166 as a failure, and then continue on with the current log.c.166 to look for the next matching oejs.AbstractConnector. If it does find its matching end log, it will be marked as a successful completion. At this point, my program was done.

Accomplished:

Properly parsing a massive file.

Learned:

- How to use regular expression matching to find key strings
- How to parse a massive file

Key Algorithms/Data Structures/OO Designs:

N/A

Uncompleted/Not Working Features:

Everything for this project was completed, and works.

Output: (evidence the code ran)

```
=== Device boot ===
31063(device5_intouch.log): 2014-01-26 09:55:07 Boot Start
31176(device5_intouch.log): 2014-01-26 09:58:04 Boot Completed
    Boot Time: 177000ms

=== Device boot ===
31274(device5_intouch.log): 2014-01-26 12:15:18 Boot Start
**** Incomplete boot ****

=== Device boot ===
31293(device5_intouch.log): 2014-01-26 14:02:39 Boot Start
31401(device5_intouch.log): 2014-01-26 14:05:24 Boot Completed
    Boot Time: 165000ms

=== Device boot ===
32623(device5_intouch.log): 2014-01-27 12:27:55 Boot Start
**** Incomplete boot ****
```

Makefile

```
EXE_NAME = ps7
OBJ = main.o
LIBS = -lboost_regex -lboost_date_time
TAR_NAME = PS7_JustinNguyen.tar.gz
CC = g++
CPPV = c++17
#DEBUG = -g -ggdb3 -fmax-errors=1
FLAGS = -Wall -Werror -pedantic -std=$(CPPV) $(DEBUG)

LL = cpplint
LFLAGS = --filter=-runtime/references --filter=-build/$(CPPV)

VFLAGS = --leakcheck=full

all: $(EXE_NAME)

$(EXE_NAME): $(OBJ)
    $(CC) $(FLAGS) -o $@ $^ $(LIBS)

%.o: %.cpp %.hpp $(DEP)
    $(CC) $(FLAGS) -c $<
%.o: %.cpp $(DEP)
    $(CC) $(FLAGS) -c $<

run: # shortcut for you
    ./$(EXE_NAME) logs/device5_intouch.log

lint:
    $(LL) $(LFLAGS) *.*pp
valgrind:
    valgrind $(VFLAGS) ./$(EXE_NAME)

dist:
    tar --exclude='.vscode' -czvf ../$(TAR_NAME) .
    mv ../$(TAR_NAME) .
clean:
    rm -f $(OBJ)
distclean: clean
    rm -f $(EXE_NAME) $(TAR_NAME)
```

main.cpp

```
// Copyright 2022 Justin Nguyen
#include <vector>
#include <string>

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

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

using std::string,
boost::regex,
boost::gregorian::date,
boost::posix_time::ptime,
boost::posix_time::time_duration;

// FORMATTING
ptime getDateFrom(const string&);
string parenthesize(const string& str) {return "(" + str + ")";}
string generateLog(unsigned int, string, ptime, bool);

// OUTPUT CONSTANTS
const char TITLE[20] = "=== Device boot ===";
const char INCOMPLETE[26] = "**** Incomplete boot ****";
const char BTIME[13] = "\tBoot Time: ";

int main(int argc, char** argv) {
    unsigned int lineNumber{0}, logStartNo;
    string fileName(argv[1]);

    ptime logStart, logEnd;  // start and end timestamps
    bool startFound(false);  // boolean to match start and end string

    std::fstream infile(fileName, std::fstream::in);
    std::fstream outfile(fileName + ".rpt", std::fstream::out);

    // match date
    regex rgxDate("^\\d{4}-[\\d{2}\\-?]+\\s?[\\d{2}:]+\\s?");
    // match log.c.166
    regex rgxStart(rgxDate.str() + "\\(log.c.166\\) server started\\s?$");
    // match oejs.AbstractConnector
    regex rgxEnd(rgxDate.str() + "\\.\\d+" + ":INFO:oejs\\.AbstractConnector:Started SelectC
```

```
45      string line;
46      while (std::getline(infile, line)) {
47          lineNumber++;
48          if (boost::regex_match(line, rgxStart)) {  // log.c.166
49              if (startFound) {  // case: start doesnt find its matching end
50                  startFound = false;
51
52                  outfile << TITLE << std::endl;
53                  outfile << generateLog(logStartNo, fileName, logStart, true)
54                  << std::endl;
55
56                  outfile << INCOMPLETE
57                  << std::endl << std::endl;
58              }
59              logStartNo = lineNumber;
60              logStart = getDateFrom(line);
61              startFound = true;
62          }
63          if (boost::regex_match(line, rgxEnd)) {  // oejs.AbstractConnector
64              startFound = false;
65              logEnd = getDateFrom(line);
66
67              outfile << TITLE << std::endl;
68
69              outfile << generateLog(logStartNo, fileName, logStart, true)
70              << std::endl;
71
72              outfile << generateLog(lineNumber, fileName, logEnd, false)
73              << std::endl;
74
75              outfile << BTIME
76              << time_duration(logEnd - logStart).total_milliseconds()
77              << "ms" << std::endl << std::endl;
78          }
79      }
80      return 0;
81  }
82
83  ptime getDateFrom(const string& dateStr) {
84      return boost::posix_time::time_from_string(
85          string(dateStr.begin(), dateStr.begin() + 19));
86  }
87
88  string generateLog(
89  unsigned int lineNo, string fileName, ptime timestamp, bool isStart) {
90      using boost::posix_time::to_iso_extended_string;
```

```
91
92      string status = isStart ? " Boot Start" : " Boot Completed";
93      std::stringstream ss;
94      string formattedTS(to_iso_extended_string(timestamp));
95
96      formattedTS.at(formattedTS.find('T')) = ' ';
97
98      ss << lineNo
99      << parenthesize(fileName) << ": "
100     << formattedTS
101     << status;
102     return ss.str();
103 }
```