Ethan Sourn

Comp IV: Project Portfolio

Spring 2021

**Contents**

1. **PS0 – Hello World with SFML 5**
   1. Overview . . . . . . . . . . 5
   2. Key Concepts . . . . . . . . . 5
   3. Learnings . . . . . . . . . . 5
   4. Output . . . . . . . . . . 5
   5. Code . . . . . . . . . . . 5
      1. Makefile . . . . . . . . . 6
      2. Main.cpp . . . . . . . . . 6
2. **PS1A – Linear Feedback Shift Register (part A) 7**
   1. Overview . . . . . . . . . . 7
   2. Key Concepts . . . . . . . . . 7
   3. Learnings . . . . . . . . . . 7
   4. Output . . . . . . . . . . 7
   5. Code . . . . . . . . . . . 7
      1. Test.cpp . . . . . . . . . 7
3. **PS1B – Linear Feedback Shift Register (part B) 9**
   1. Overview . . . . . . . . . . 9
   2. Key Concepts . . . . . . . . . 9
   3. Learnings . . . . . . . . . . 9
   4. Output . . . . . . . . . . 9
   5. Code . . . . . . . . . . . 10
      1. Makefile . . . . . . . . . 10
      2. PhotoMagic.cpp . . . . . . . . 11
      3. FibLFSR.cpp . . . . . . . . . 12
      4. FibLFSR.h . . . . . . . . . 12
4. **PS2A – N-Body Simulation (part A) 13**
   1. Overview . . . . . . . . . . 13
   2. Key Concepts . . . . . . . . . 13
   3. Learnings . . . . . . . . . . 13
   4. Output . . . . . . . . . . 14
5. **PS2B – N-Body Simulation (part B) 14**
   1. Overview . . . . . . . . . . 14
   2. Key Concepts . . . . . . . . . 14
   3. Learnings . . . . . . . . . . 14
   4. Output . . . . . . . . . . 15
   5. Code . . . . . . . . . . . 16
      1. Makefile . . . . . . . . . 16
      2. Main.cpp . . . . . . . . . 16
      3. NBody.cpp . . . . . . . . . 19
      4. NBody.h . . . . . . . . . 22
6. **PS3 – DNA Sequence Alignment 23**
   1. Overview . . . . . . . . . . 23
   2. Key Concepts . . . . . . . . . 23
   3. Learnings . . . . . . . . . . 23
   4. Output . . . . . . . . . . 23
   5. Code . . . . . . . . . . . 24
      1. Makefile . . . . . . . . . 24
      2. Main.cpp . . . . . . . . . 24
      3. ED.cpp . . . . . . . . . 24
      4. ED.h . . . . . . . . . . 26
7. **PS4A – Synthesizing a Plucked String Sound (part A) 27**
   1. Overview . . . . . . . . . . 27
   2. Key Concepts . . . . . . . . . 27
   3. Learnings . . . . . . . . . . 27
   4. Output . . . . . . . . . . 28
8. **PS4B – Synthesizing a Plucked String Sound (part B) 28**
   1. Overview . . . . . . . . . . 28
   2. Key Concepts . . . . . . . . . 28
   3. Learnings . . . . . . . . . . 28
   4. Output . . . . . . . . . . 28
   5. Code . . . . . . . . . . . 29
      1. Makefile . . . . . . . . . 29
      2. KSGuitarSim.cpp . . . . . . . . 29
      3. CircularBuffer.cpp . . . . . . . . 31
      4. StringSound.cpp . . . . . . . . 33
      5. CircularBuffer.h . . . . . . . . 35
      6. StringSound.h . . . . . . . . . 36
9. **PS5 – Recursive Graphics 37**
   1. Overview . . . . . . . . . . 37
   2. Key Concepts . . . . . . . . . 37
   3. Learnings . . . . . . . . . . 38
   4. Output . . . . . . . . . . 38
   5. Code . . . . . . . . . . . 38
      1. Makefile . . . . . . . . . 38
      2. TFractal.cpp . . . . . . . . . 38
      3. Triangle.cpp . . . . . . . . . 38
      4. Triangle.h . . . . . . . . . 38
10. **PS6 – Kronos – Intro to Regular Expression 38**
    1. Overview . . . . . . . . . 38
    2. Key Concepts . . . . . . . . . 39
    3. Learnings . . . . . . . . . 39
    4. Output . . . . . . . . . . 39
    5. Code . . . . . . . . . . 41
       1. Makefile . . . . . . . . . 41
       2. kronos.cpp . . . . . . . . . 42

Time to complete: 5 ~ 6 hrs

1. **PS0 – Hello World with SFML**
   1. **Overview**

The main purpose of this assignment is to set up and get running the correct build environment. Afterwards, we needed to build the SFML Hello World demo code and extend it with several new features. These new features such as drawing an image sprite, making it move, having it respond to keystrokes, and something else up to us. Finally, we needed to create a Makefile to compile the program with the flags “-Wall -Werror and -pedantic”. Which will be a requirement for every assignment hence forth.

I completed every part of the assignment. I created the Makefile, got the build environment and demo code working. I then extended the demo code and created a simple green circle sprite and programmed it to move across the screen in response to pressing the A, S, D, and W keys. For the fourth requirement I had the sprite rotate clockwise and counterclockwise in respect to the J and K keys. When I was done, I took a screenshot of the SFML window and created a gzipped tar archive of the directory. Which along with the Makefile will be the way we turn in every assignment.

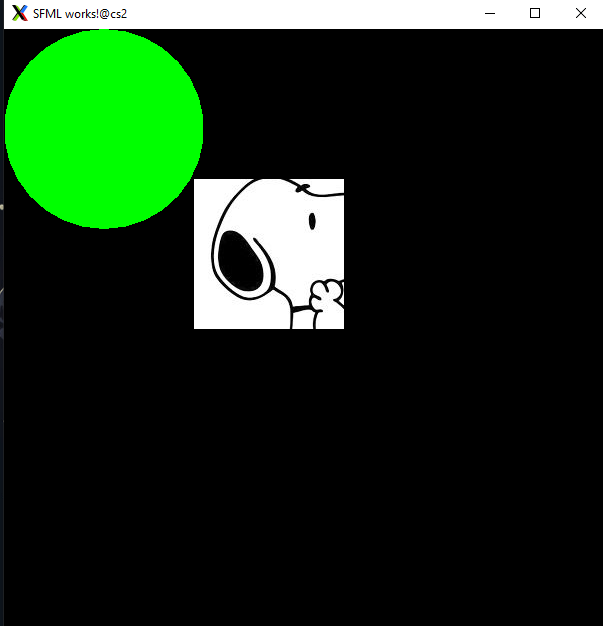
* 1. **Key Concepts**

Since this was an introductory assignment, there were no key algorithms, data structures, or object-oriented designs that were central to the assignment.

* 1. **Learnings**

I learned more about the Simple and Fast Multimedia Library (SFML) interface and some of its API documentation relating to working with graphics. Such as creating a sprite, which is a just an entity given a texture.

* 1. **Output**



* 1. **Code**
     1. **Makefile**

****

* + 1. **Main.cpp**



****

1. **PS1A – Linear Feedback Shift Register (part A)**
   1. **Overview**

In this assignment we were tasked to write a program that produces pseudo-random bits by simulating a linear feedback shift register, and then use it to implement a simple form of encryption for digital pictures. We needed to implement the FibLFSR class API which gave us a constructor and two functions for us to complete. Alongside using unit tests from the Boost test framework to make sure our code is correct. The step function simulates one step of the LFSR and returns the rightmost bit as an integer (0 or 1). The generate function takes an integer k as ana argument and returns a k-bit integer obtained by simulating k steps of the LFSR.

I finished the assignment. I completed the FibLFSR class and created a separate file called Test.cpp that used the Boost unit tests. It helped me evaluate if my math was correct by allowing test cases to be made that verified if my functions were working. For example, given a specific seed I would test if after a certain number of steps against the correct value. If it were wrong, it would return an error. So, using the unit tests I was able to evaluate every aspect of the assignment to have it properly work.

* 1. **Key Concepts**

This assignment utilized core object-oriented programming concepts such as objects and classes. We were given a starter class and a small text description of what it should accomplish and return. We needed to choose the internal representation, implement the constructor, and the three member functions.

* 1. **Learnings**

I learned about linear feedback shift registers and how they are implemented. The concepts of seeds, XOR operations, and incremental unit testing. It is a great way of testing code to make sure the right output is correct.

* 1. **Output**



* 1. **Code** 
     1. **Test.cpp**



****

1. **PS1B – Linear Feedback Shift Register (part B)**
   1. **Overview**

This assignment is a continuation of the previous assignment. Here we are extending the code to read three arguments from the command line, use SFML to load the source image from disk and display it in its own window. By using the debugged FibLFSR class we encode or decode the source image with a randomized set of numbers that affect the color of the pixels through a transform function.

I managed to complete the project and implement the transform function. I created a new file called PhotoMagic which handled all the SFML contents. Such as generating the loop that handled the graphics and the SFML window. It was also where I created the sprites that would in turn have each of its pixel’s randomized by my transform function. Finally, the new image would be saved to a new PNG file.

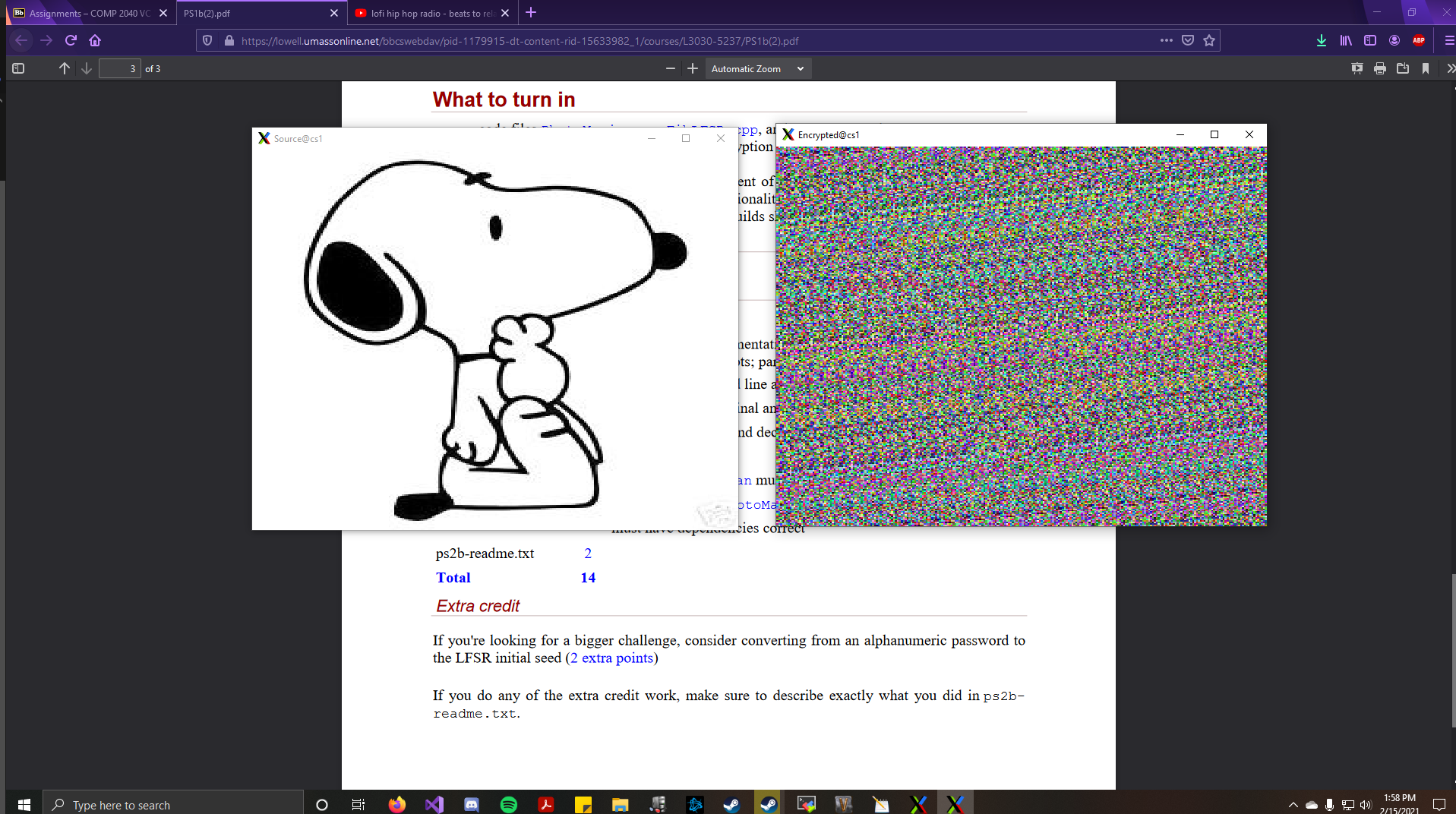
* 1. **Key Concepts**

In this assignment we worked with classes and objects. One central object-oriented programming concept was abstraction. I declared public and private members along with access specifiers to show only essential code.

* 1. **Learnings**

This assignment further helped my understanding with the SFML library and practice with coding in C++. There were no new concepts to be learned but rather this assignment was the first that I found very interesting and somewhat practical. Encrypting and decrypting pictures through a linear feedback register.

* 1. **Output**



* 1. **Code**
     1. **Makefile**

****

* + 1. **PhotoMagic.cpp**

****

****

* + 1. **FibLFSR.cpp**

****

* + 1. **FibLFSR.h**

****

1. **PS2A – N-Body Simulation (part A)**
   1. **Overview**

For this assignment we were tasked to simulate the motions of N particles in a plane, mutually affected by gravitational forces, and animate the results. Specifically, for part A of this assignment, we must load and display a static universe. Our program needed to be able to read and download universe specification files and image files from a given zip file. Then it should be able to read the file through the command line with input redirection from a text file. The text file contained all the parameters for the particles in space, such as its x and y position, velocity, mass, and images.

I completed the assignment by implementing two classes called the CelestialBody and Universe. The Universe class encompasses the CelestialBody and so it reads the parameters from the text file. Then it creates a vector of pointers to CelestialBody objects that I instantiated with the keyword new. The CelestialBody class then sets the private variables, creates sprites for each object, and perform the math for the position of the sprites.

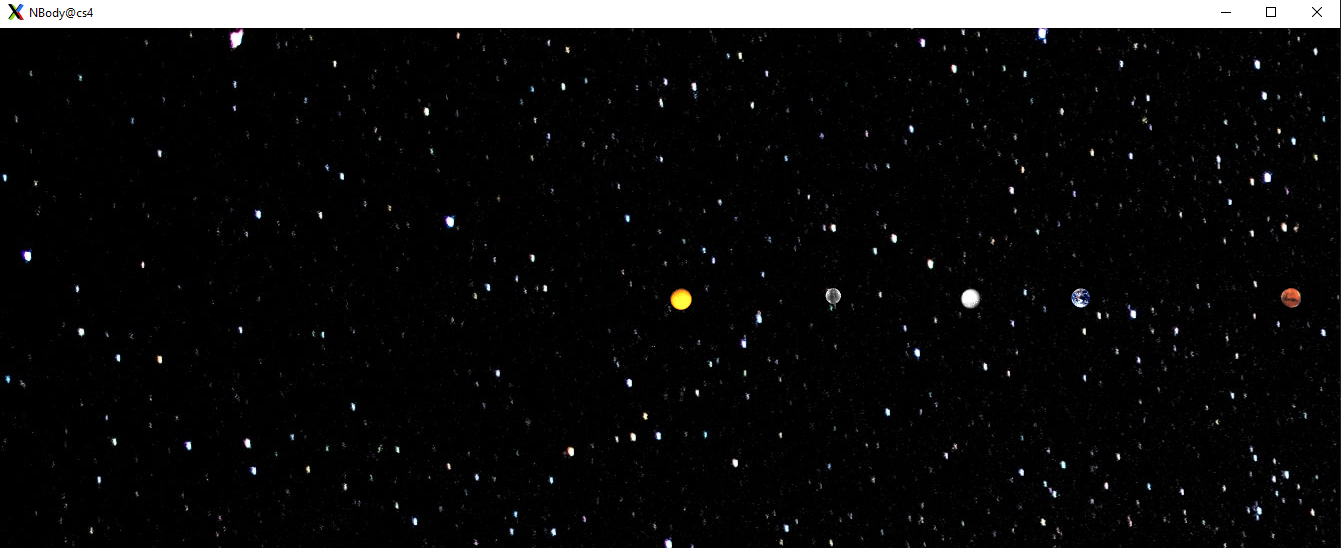
* 1. **Key Concepts**

One key data structure that was integral to the assignment was the vector container. It helped me store all my class objects and access their respective parameters to complete the assignment. I thought it was the most straightforward way of implementing and constantly updating the particles respective parameters.

* 1. **Learnings**

I learned how to have my program read input from the command line and display multiple images and sprites.

* 1. **Output**

****

1. **PS2B – N-Body Simulation (part B)** 
   1. **Overview**

This assignment is a continuation of the previous and here we add physics to the simulation to simulate our inner solar system’s orbit around the sun. We need to extend our CelestialBody class with mutators so that we can modify the velocities of each object, implement a method called step which takes a time parameter, and moves the entity given its internal velocity for that much time. The main routine of our program should keep track of elapsed time and terminate the simulation after a certain amount of time has passed provided at the command line. After the animation stops, it outputs the final state of the universe as a table.

This was the first assignment I had some trouble on. My planets immediately fly off the screen upon running the program. Which lead me to believe either my math for calculating pairwise forces is wrong, or I am not correctly updating the velocities and position. Otherwise, I completed several of the implementation requirements such as using smart pointers, displaying elapsed time, and playing a sound file.

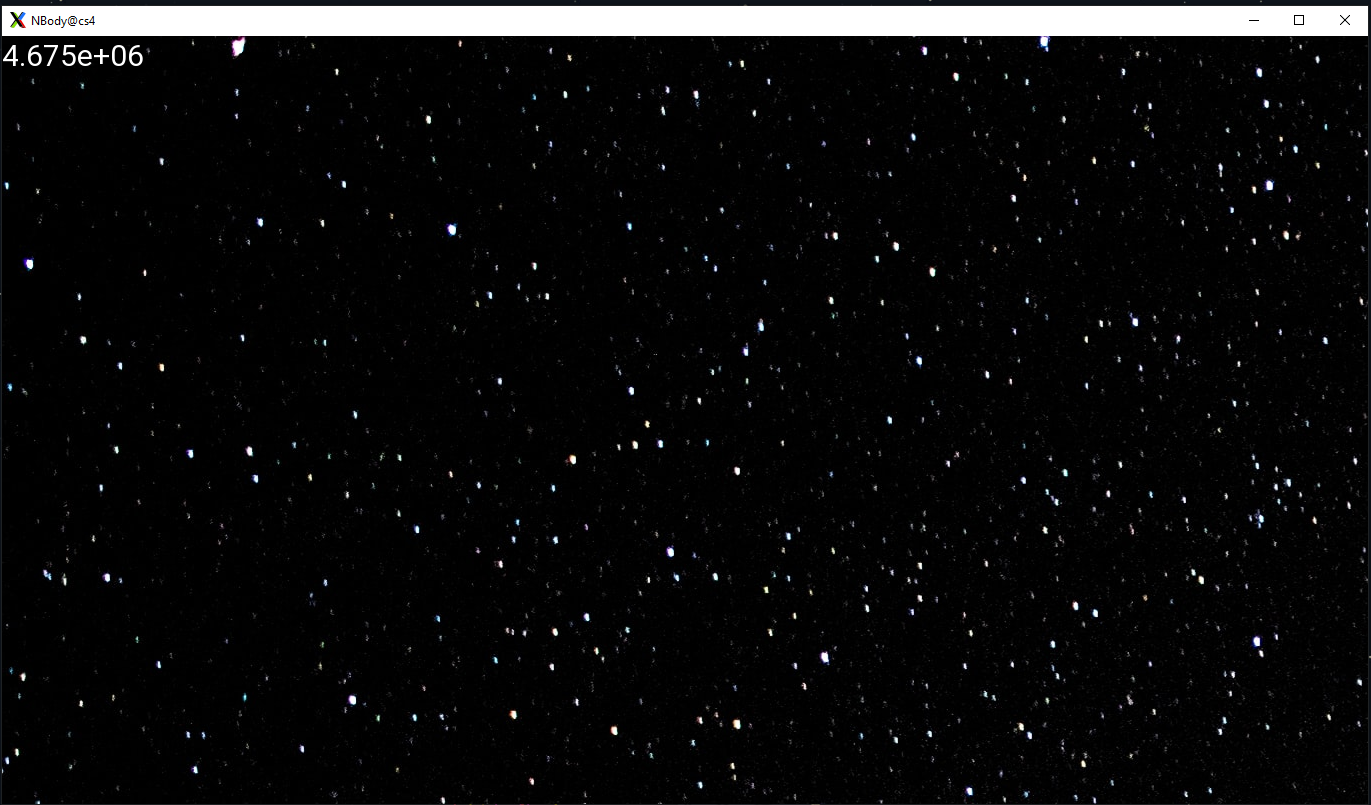
* 1. **Key Concepts**

One important concept for this assignment was encapsulation. I had two classes in my program that relied on each other and utilized each other’s functions. I made sure to have each of my classes interact in such a way only the essential data is shared. Private members operated independently in their own classes and mutators were made to access such.

* 1. **Learnings**

I learned how to utilize the smart pointer class which alleviates the many problems with regular pointers such as leaking memory and remembering to delete the object it is assigned to. The automatic memory management is very helpful and made the assignment easier to work on. This assignment also taught me how to calculate pair wise forces, the net force, and acceleration of celestial bodies.

* 1. **Output**

****

****

* 1. **Code**
     1. **Makefile**



* + 1. **Main.cpp**

****

****

* + 1. **NBody.cpp**

****

****

****

****

* + 1. **NBody.h**

****

****

1. **PS3 – DNA Sequence Alignment**
   1. **Overview**

For this assignment we are task to solve a fundamental problem in computational biology and to learn about the dynamic programming paradigm. Given two random gene sequences, we need to calculate the edit-distance between each character from the sequences. If two characters mismatch, we produce a numerical score of 1 that refers to the “cost”. If the two sequences are not the same length, we need to insert a gap which has a cost of 2. If they match, they have a cost of 0. The goal is to calculate the best alignment (the lowest total cost/edit-distance) between two genetic sequences over all possible alignments. The genetic sequences were given in a folder filled with text files which we had to read and correctly compare.

There were many ways to approach implementation, I chose to go with dynamic programming using an NxM matrix, called the Needleman and Wursh method. I get to avoid recursion and this methodology is more intuitive for me to think through. I completed the assignment and created the class ED that has a constructor and four additional functions that helped me calculate the edit-distance.

* 1. **Key** **Concepts**

The dynamic programming paradigm is an algorithmic approach that was integral to completing this assignment. It enabled me to break apart the assignment into smaller sub-problems.

* 1. **Learnings**

As stated above, I learned about the dynamic programming paradigm and the concept of computational biology.

* 1. **Output**

****

* 1. **Code**
     1. **Makefile**



* + 1. **Main**.**cpp**



* + 1. **ED**.**cpp**





* + 1. **ED**.**h**



1. **PS4A – Synthesizing a Plucked String Sound (part A)**
   1. **Overview**

For this assignment, we are tasked to simulate plucking a guitar string using the Karplus-Strong algorithm. Alongside practicing good coding habits with Cpplint style guidelines, utilizing unit testing and exceptions. We are given a class named CircularBuffer that implements an API that has a constructor and 6 different functions that all maintain a buffer feedback mechanism. The ring buffer holds N samples that are scaled by an energy decay factor that simulates a frequency. The buffer should then be able to add and remove samples so that it can continuously take in input and produce sound. However, part A only requires us to complete the class and not actually implement the SFML components to generate the notes.

I completed the CircularBuffer class and tested it with a whole suite of Boost test cases. Each test case pertains to each function and ensures that everything is working, and the output is correct. I also implemented several exceptions that worked with the test cases, allowing me to report to myself specific messages when something does not work. Below in the Output section you will noticed that there are four error messages and a status message from Boost stating that there are no errors. The error messages were intentional to test my functions and constructor. Finally, I tested all my code against Cpplint style guidelines for proper formatting.

* 1. **Key** **Concepts**

Exception handling was an integral part of this assignment. It provided my program a way to react to exceptional circumstances like runtime errors and invalid arguments.

* 1. **Learnings**

This was the first assignment that required us to use Cpplint formatting for our code. It taught me a lot on what was seen as acceptable coding practice. Such as excluding certain libraries that are redundant and/or bloated, using spaces instead of tabs, etc.

* 1. **Output**



1. **PS4B – Synthesizing a Plucked String Sound (part B)**
   1. **Overview**

PS4B is a continuation of the previous assignment, but here we are using the SFML library to produce sound from the window. It introduces a new class called StringSound that performs the math related to the Karplus-Strong string simulation described previously. Our program needed to support a total of 37 notes on the chromatic scale from 100Hz to 880Hz. Those notes corresponded to the following keys form lowest note to highest: “q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/'”. We were also given some starter code that helped generate the audio in SFML.

I completed the assignment and got some output to show on screen. However, I never managed to hear any sound and I think the issue was that I was writing this program on MobaXterm. Many of my peers ran into the same issue. Regardless, I completed the StringSound class, managed to open the SFML window and display some output that corresponds to the keyboard.

* 1. **Key** **Concepts**

My approach to this assignment required the use a vector container. It simulated the circular buffer. I was able to continuously add elements, remove them, and report the status of the vector i.e., if it is full or empty, the current number of items, and what was at the front of the buffer.

* 1. **Learnings**

I learned how to generate audio with the SFML library. I also learned how to simulate the notes on a keyboard. With that knowledge I can probably make my program produce a different sound and mimic a different instrument.

* 1. **Output**



* 1. **Code**
     1. **Makefile**



* + 1. **KSGuitarSim**.**cpp**





* + 1. **CircularBuffer**.**cpp**





* + 1. **StringSound**.**cpp**





* + 1. **CircularBuffer**.**h**



* + 1. **StringSound**.**h**



1. **PS5 – Recursive Graphics (Triangle Fractal)**
   1. **Overview**

In this assignment we needed to write a program that plots a triangle fractal, a variation of the Sierpinski triangle. Our program needed to implement a recursive function called fTree(). And a main program that calls the recursive function. It needed to take two command-line arguments that correspond to the length of the base equilateral triangle and the depth of the recursion. We also needed to implement a class called Triangle which derives from sf::Drawable.

Unfortunately, while I was working on this assignment, I ran into a fatal error that came from the server. It stated that there was no space left on the device and it deleted all the contents of my code except for the Makefile. This section will be updated soon with my completed code and appropriate output

* 1. **Key** **Concepts**

Derived classes and inheritance played an integral role in completing this assignment. In order to have any sort of output without recreating the class, I needed to inherit from a class that had all the appropriate functionality completed.

* 1. **Learnings**

I learned how to recursively implement a triangle fractal tree.

* 1. **Output**

NOT FINISHED

* 1. **Code**
     1. **Makefile**



* + 1. **TFractal**.**cpp**

NOT FINISHED

* + 1. **Triangle**.**cpp**

NOT FINISHED

* + 1. **Triangle**.**h**

NOT FINISHED

1. **PS6 – Kronos – Intro to Regular Expression**
   1. **Overview**

For this assignment we were tasked to analyze the Kronos InTouch time clock’s log by using regular expressions to parse the file. The InTouch device is a Linux clock that stored information of its own status. If the device failed to boot or succeeds, log files are created to help distinguish it.

I believe I completed the project correctly. I used the required Boost regex library to create several expressions that essentially searched the entire InTouch log for specific phrases/entries. When the device first boots up, there is a logging message that indicates the server started and is paired with another specific log entry. Therefore, there are specific message sequences that indicates a failed or successful boot, and I used the many functions and classes in the Boost regex library to identify them.

* 1. **Key** **Concepts**

Regular expressions were the focus of the assignment. The Boost regex library provided two important classes: “boost::regex” and “boost::smatch” which the former defines a regular expression, and the latter saves the results from the search. The function “boost::regex\_search()” made searching a string for a regular expression simple.

* 1. **Learnings**

I learned about regular expressions which help with pattern searching against sequences of characters. It was very confusing at first especially with the syntax when defining a regular expression.

* 1. **Output**





* 1. **Code**
     1. **Makefile**

****

* + 1. **kronos**.**cpp**



