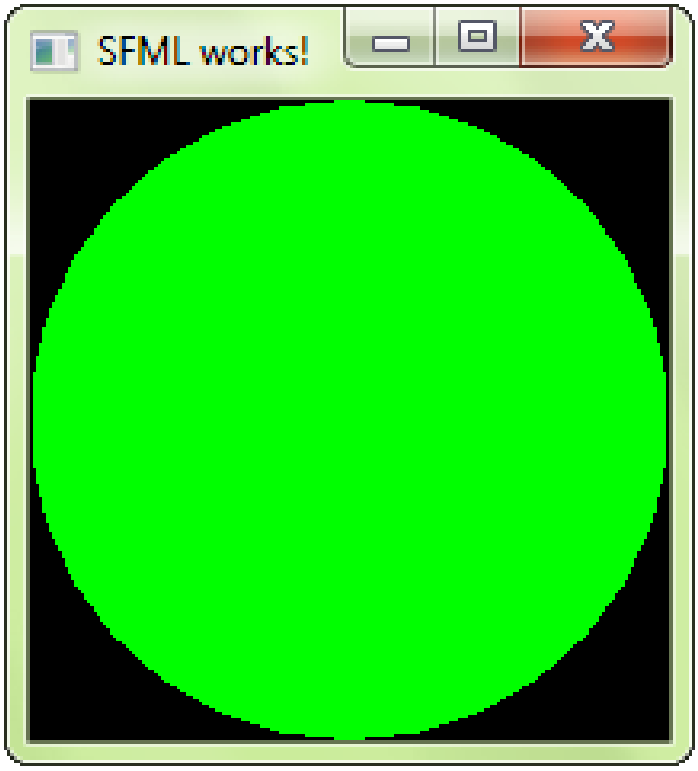
Jacob Blumsack
COMP IV: Project Portfolio
Fall 202

Table of Contents

PSO

Hello World with SFML



PS0: Makefile

```
1. CC = g++
2. CFLAGS = -c -g -Og -Wall -Werror -ansi -pedantic
3. OBJ = main.o
4. DEPS =
5. LIBS = -lsfml-graphics -lsfml-window -lsfml-system
6. EXE = SFML-app
7.
8. all: $(OBJ)
9.         $(CC) $(OBJ) -o $(EXE) $(LIBS)
10.
11.     %.o: %.cpp $(DEPS)
12.             $(CC) $(CFLAGS) -o $@ $<
13.
14.     clean:
15.             rm $(OBJ) $(EXE)
```

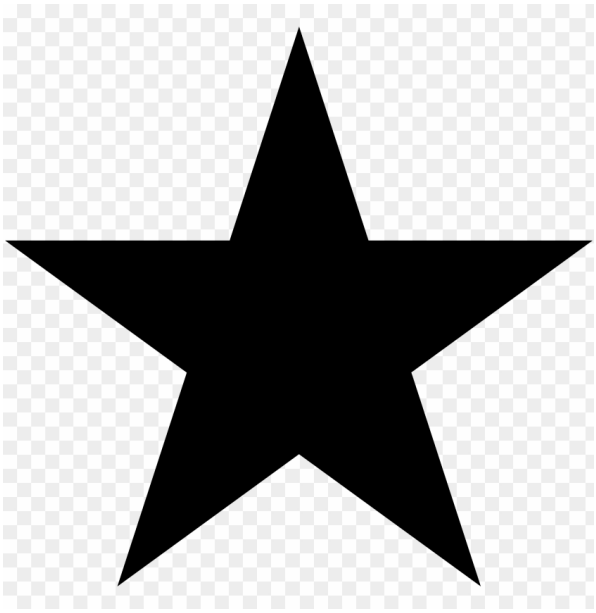
PS0: main.cpp

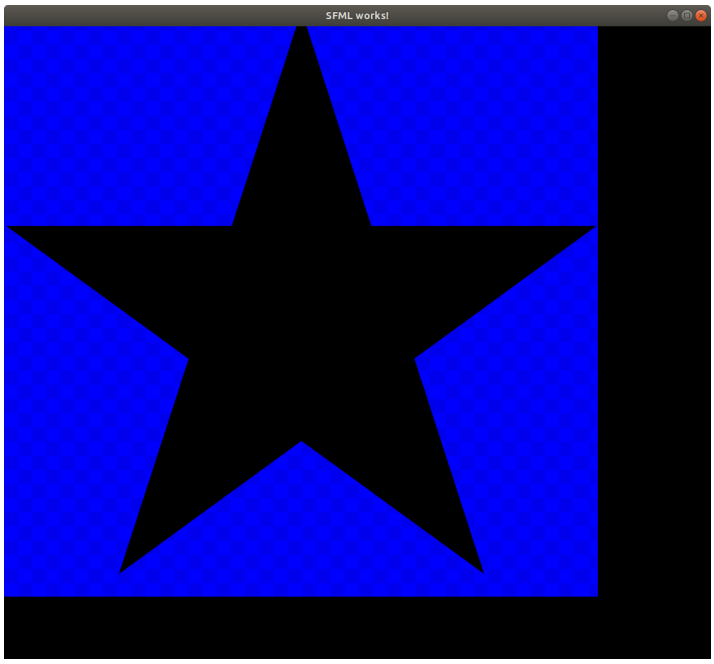
```
1. #include <SFML/Graphics.hpp>
2. #include <iostream>
3.
4. #define SPRITE_SPEED 5
5.
6. int main()
7. {
8.   sf::RenderWindow window(sf::VideoMode(1000, 1000), "SFML
   works!");//create window
9.     window.setVerticalSyncEnabled(true);
10.         window.setKeyRepeatEnabled(false);
11.
12.         sf::Texture texture;//create texture
13.
14.         int x = window.getSize().x/2.;//initialize
   movement coordinates
15.         int y = window.getSize().y/2.;
16.
17.         int i = 0; //initialize color variables
18.         int j = 0;
19.         int k = 255;
20.
21.         bool up = false; //initialize movement variables
   for keystrokes
22.         bool down = false;
23.         bool left = false;
24.         bool right = false;
25.
26.         if (!texture.loadFromFile("Star.png"))//load
   sprite
27.           {
28.           return EXIT_FAILURE;
29.           }
30.
31.         sf::Sprite sprite(texture);
32.         sprite.setOrigin(500,500);//set origin
33.
34.         sf::Clock timer;
35.         while (window.isOpen())//open window
36.           {
37.          sf::Event event;
38.          while (window.pollEvent(event))
39.            {
```

```
40.             if (event.type == sf::Event::Closed)
41.               {
42.                window.close();
43.               }
44.             if (event.type == sf::Event::KeyPressed)
45.               {
46.                switch (event.key.code)
47.                 {
48.                 case sf::Keyboard::Escape: window.close();
   break;//use wasd for movement and esc to leave window
49.                 case sf::Keyboard::W: up = true; break;
50.                 case sf::Keyboard::S: down = true; break;
51.                 case sf::Keyboard::A: left = true; break;
52.                 case sf::Keyboard::D: right = true; break;
53.                 default: break;
54.                 }
55.               }
56.             if (event.type == sf::Event::KeyReleased)
57.               {
58.                switch (event.key.code)
59.                 {
60.                 case sf::Keyboard::W: up = false;
   break;//shut off movement when keys are released
61.                 case sf::Keyboard::S: down = false; break;
62.                 case sf::Keyboard::A: left = false; break;
63.                 case sf::Keyboard::D: right = false;
   break;
64.                 default: break;
65.                 }
66.               }
67.             if
   (sf::Keyboard::isKeyPressed(sf::Keyboard::R))//set sprite
   color for red
68.               {
69.                i = 255;
70.                j = 0;
71.                k = 0;
72.               }
73.             if
   (sf::Keyboard::isKeyPressed(sf::Keyboard::G))//set sprite
   color for green
74.               {
75.                i = 0;
76.                j = 255;
77.                k = 0;
78.               }
```

```
79.              if
   (sf::Keyboard::isKeyPressed(sf::Keyboard::B))//set sprite
   color for blue
80.                {
81.               i = 0;
82.               j = 0;
83.               k = 255;
84.                }
85.           }
86.         if (up)
87.           {
88.             y = y - SPRITE_SPEED;
89.           }//update sprite movement
90.         if (down)
91.           {
92.             y = y + SPRITE_SPEED;
93.           }
94.         if (left)
95.           {
96.             x = x - SPRITE_SPEED;
97.           }
98.         if (right)
99.           {
100.            x = x + SPRITE_SPEED;
101.          }
102.
103.        sprite.setPosition(x,y);
104.        sprite.setColor(sf::Color(i,j,k));
105.        window.clear();
106.        window.draw(sprite);
107.        window.display();
108.         }
109.       return EXIT_SUCCESS;
110.   }
```

PS0: Output





PS0: Discussion

What was Accomplished:

Being the first programming assignment of the class, the one of the goals of this assignment was to initialize and setup the virtual environment (virtualbox) that we would be programming in for the rest of the semester. Another goal was to setup the Simple Fast Media Library (SFML) for programming as it would be used in future assignments as well. Once the initial setup was complete, the actual programming goal of the assignment was to display an SFML window, load and display a sprite, and use keystrokes to move the sprite. For the implementation of movement, the W, A, S, and D keys were used for up, left, down, and right respectively. This was all accomplished in the assignment as instructed and an additional feature was also programmed. This additional feature was the addition of the ability to change the color of the sprite using the R, G, and B keys to toggle between red, green, and blue respectively. In my case, the sprite I chose was of a star as seen in the output section. The output page shows the star sprite that was used. The second image demonstrates how the star could move and change color as described.

Key Algorithms, Data Structures, OO Designs:

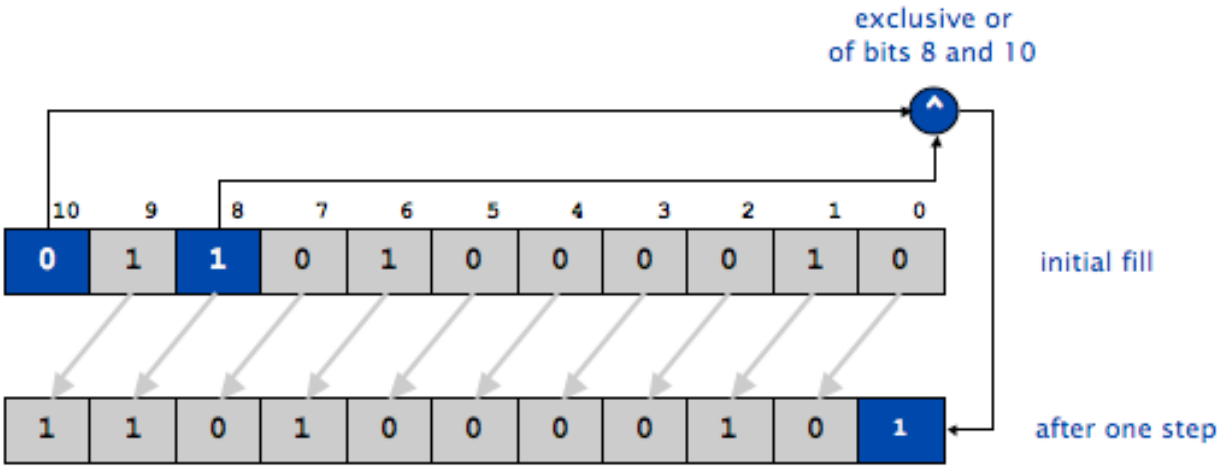
This assignment mainly focused on experimenting with the SFML library. Having only a Makefile and a main.cpp it lacked on the front of OO Design, data structures, and key algorithms. Implementation wise, switch statements were used for toggling between W, A, S, and D for the movement of the sprite. If statements were used to switch between each color change. To change color, on the press of any R, G, or B key the maximum value for red, green, or blue values of the sprite would be maximized to 255 and the rest to 0. For example, if B was pressed, the red value would be 0, the green value would be 0, and the blue value would be 255.

What was Learned:

This was the first time I had used the SFML library so it was all a learning experience when it came to finding the correct built in functions to make the assignment function properly. The SFML documentation proved to be a very valuable resource. For instance, SF::Keyboard was a function that was needed as it allowed for the functionality of moving the sprite on key pressed. Other functions of the SFML library such as the SF::Sprite were absolutely crucial to the correct implementation of the assignment as without it would have been impossible to display a sprite to the window. Another important aspect that was learned during this assignment was how to work with the virtualbox virtual machine. This was crucial as well as it was used for every programming assignment of the semester so becoming familiar with it was very important. I had not used linux too often before this so it was a curve simply finding where everything I needed was on the virtual linux machine.

PS1a

Linear Feedback Shift Register



One step of an 11-bit LFSR with initial fill 01101000010 and tap at position 8

PS1a: Makefile

```
1. CC = g++
2. CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
3. OBJ = test.o
4. DEPS =
5. LIBS =
6. EXE = test
7.
8. all: ps1a
9. ps1a: FibLFSR.o test.o
10.     $(CC) FibLFSR.o test.o -o ps1a
11.     test.o: test.cpp FibLFSR.h
12.     $(CC) -c test.cpp FibLFSR.h $(CFLAGS)
13.     FibLFSR.o: FibLFSR.cpp FibLFSR.h
14.     $(CC) -c FibLFSR.cpp $(CFLAGS)
15.
16.     clean:
17.     rm $(OBJ) $(EXE)
```

PS1a: test.cpp

```
1. // Dr. Rykalova
2. // test.cpp for PS1a
3. // updated 1/31/2020
4.
5. #include <iostream>
6. #include <string>
7.
8. #include "FibLFSR.h"
9.
10.     #define BOOST_TEST_DYN_LINK
11.     #define BOOST_TEST_MODULE Main
12.     #include <boost/test/included/unit_test.hpp>
13.
14.     BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
15.
16.       FibLFSR l("1011011000110110");
17.       BOOST_REQUIRE(l.step() == 0);
18.       BOOST_REQUIRE(l.step() == 0);
19.       BOOST_REQUIRE(l.step() == 0);
20.       BOOST_REQUIRE(l.step() == 1);
21.       BOOST_REQUIRE(l.step() == 1);
22.       BOOST_REQUIRE(l.step() == 0);
23.       BOOST_REQUIRE(l.step() == 0);
24.       BOOST_REQUIRE(l.step() == 1);
25.
26.       FibLFSR l2("1011011000110110");
27.       BOOST_REQUIRE(l2.generate(9) == 51);
28.     }
29.
30.     BOOST_AUTO_TEST_CASE(ExtraTest1) {
31.       FibLFSR a1("00000001");//test another case with use
    of step()
32.       BOOST_REQUIRE(a1.step() == 0);//outcome of each step
    is checked
33.       BOOST_REQUIRE(a1.step() == 0);
34.       BOOST_REQUIRE(a1.step() == 1);
35.       BOOST_REQUIRE(a1.step() == 0);
36.       BOOST_REQUIRE(a1.step() == 1);
37.       BOOST_REQUIRE(a1.step() == 0);
38.       BOOST_REQUIRE(a1.step() == 0);
39.
40.       FibLFSR a2("00000001");//check generate() with
    another FibLFSR Object
```

```
41.       BOOST_REQUIRE(a2.generate(5) == 10);//check output
42.     }
43.
44.     BOOST_AUTO_TEST_CASE(ExtraTest2) {
45.       FibLFSR a1("11111111");//test another case with use
  of step()
46.       BOOST_REQUIRE(a1.step() == 0);//outcome of each step
  is checked
47.       BOOST_REQUIRE(a1.step() == 0);//Testing all 1 input
  for step() and generate()
48.       BOOST_REQUIRE(a1.step() == 0);
49.       BOOST_REQUIRE(a1.step() == 1);
50.       BOOST_REQUIRE(a1.step() == 1);
51.       BOOST_REQUIRE(a1.step() == 0);
52.       BOOST_REQUIRE(a1.step() == 0);
53.
54.       FibLFSR a2("11111111");//check generate() with
  another FibLFSR Object
55.       BOOST_REQUIRE(a2.generate(5) == 12);//check output
56.     }
```

PS1a: FibLFSR.h

```
1. #include <string>
2. #include <iostream>
3.
4. using namespace std;
5.
6. class FibLFSR {
7. public:
8.     FibLFSR(string seed);
9.     ~FibLFSR();
10.         int step();
11.         int generate(int k);
12.         friend ostream& operator<< (ostream &out, const
   FibLFSR& fiblfsr);
13.     private:
14.         string Seed;//stores seed
15.         int rightMost;//stores rightmost
16.     };
```

PS1a: FibLFSR.cpp

```
1. #include <iostream>
2. #include <string>
3. #include "FibLFSR.h"
4. #include <string>
5. #include <stdlib.h>
6. #include <math.h>
7.
8. using namespace std;
9. /*//main for quick testing
10.     int main() {
11.         FibLFSR l1 = FibLFSR("1011011000110110");
12.         cout << l1 << endl;
13.         l1.step();
14.         cout << l1 << endl;
15.         l1.step();
16.         cout << l1 << endl;
17.         l1.step();
18.         cout << l1 << endl;
19.         l1.step();
20.         cout << l1 << endl;
21.         l1.step();
22.         cout << l1 << endl;
23.         l1.step();
24.         cout << l1 << endl;
25.         l1.step();
26.         cout << l1 << endl;
27.         l1.step();
28.         cout << l1 << endl;
29.         l1.step();
30.         cout << l1 << endl;
31.         l1.step();
32.         cout << l1 << endl << endl;
33.
34.         cout << "TEST GENERATE" << endl;
35.         FibLFSR l2 = FibLFSR("1100011011000011");
36.         l2.generate(5);
37.         cout << l2 << endl;
38.         return 0;
39.     }
40.     */
41.     FibLFSR::FibLFSR(string seed) {
42.         Seed = seed;//initialize seed as input
43.         int a0 = Seed[0] - 48;//initialize rightmost
```

```
44.         int a2 = Seed[2] - 48;
45.         int a3 = Seed[3] - 48;
46.         int a5 = Seed[5] - 48;
47.         int xor1 = (a3 ^ a5);
48.         int xor2 = (a0 ^ a2);
49.         rightMost = xor1 ^ xor2;
50.     }
51.
52.     FibLFSR::~FibLFSR() {
53.         delete &Seed;
54.     }
55.
56.     ostream& operator<< (ostream& out, const FibLFSR&
    fiblfsr) {
57.         out << fiblfsr.Seed << " " << fiblfsr.rightMost <<
    endl; //print seed then rightmost
58.         return out;
59.     }
60.
61.     int FibLFSR::step() {
62.         int x0 = Seed[0] - 48;//cast chars from string to
    ints for xor
63.         int x2 = Seed[2] - 48;
64.         int x3 = Seed[3] - 48;
65.         int x5 = Seed[5] - 48;
66.         int xor1 = (x3 ^ x5);//xor by fibonacci pattern
67.         int xor2 = (x0 ^ x2);
68.         rightMost = xor1 ^ xor2;
69.         char xorChar = rightMost + 48;
70.         Seed.erase(0, 1);//erase first bit (shift)
71.         Seed = Seed + xorChar;//add next character to end
    of seed
72.         return rightMost;//update rightmost
73.
74.     }
75.
76.
77.     int FibLFSR::generate(int k) {
78.         int var1 = 0;
79.         for (int z = 0; z != k; z++) {
80.             int d = step();//call step function k times
81.             var1 += (d * pow(2, (z - 1)));//convert binary
    to decimal
82.         }
83.         rightMost = var1;  //update rightmost
84.         return rightMost; //return generated output
85.     }
```

PS1a: BoostTest.cpp

```
1. #define BOOST_TEST_MODULE stringtest
2. #include <boost/test/included/unit_test.hpp>
3. #include "./str.h"
4.
5. BOOST_AUTO_TEST_SUITE (stringtest) // name of the test
   suite is stringtest
6.
7. BOOST_AUTO_TEST_CASE (test1)
8. {
9.   mystring s;
10.       BOOST_CHECK(s.size() == 1);
11.     }
12.
13.     BOOST_AUTO_TEST_CASE (test2)
14.     {
15.       char msg[] = "Hello world";
16.       mystring s;
17.       s.setbuffer(msg);
18.       BOOST_REQUIRE_EQUAL ('h', s[0]); // basic test
19.     }
20.
21.     BOOST_AUTO_TEST_SUITE_END( )
```

PS1a: Output

```
1011011000110110 0

0110110001101100 0

1101100011011000 0

1011000110110000 0

0110001101100001 1

1100011011000011 1

1000110110000110 0

0001101100001100 0

0011011000011001 1

0110110000110011 1

1101100001100110 0

TEST GENERATE
1101100001100110 6
```

PS1a: Discussion

What was Accomplished:

In this assignment, the implementation of a linear feedback shift register (LSFR) based on the Fibonacci sequence was accomplished. This register generates a semi random number based off shifting an N bit long binary seed string by exclusive or-ing at specified tap positions. This was accomplished by implementing first a constructor to initialize the seed string and the tap. A step function was created to perform the exclusive or-ing once and step all the bits to the left once and place the new bit at the farthest right position. The step locations were based on the Fibonacci sequence hence the name “Fibonacci LFSR”. Finally, a generate function was made to run the step function k times and return the new string that was produced. Test files were also implemented to ensure that the class functions were working as expected. The output page shows a sample main I wrote to display the step and generate functions in action.

Key Algorithms, Data Structures, OO Designs:

This assignment required the creation of a FibLFSR class. This would allow the formation of FibLFSR objects that could input a string and allow for the implementation of the Fibonacci LFSR upon the inputted seed string. Encapsualtion and abstraction were practiced in this assignment as there were private variables and public functions that could access these variables. For instance, the constructor was implemented for the sake of initializing the private variables of the class object. Polymorphism was also demonstrated through the overloading of the insertion operator. Polymorphism is the idea something can have different forms under different circumstances. In this case the stream insertion operator was overloaded to add new functionality to display the current register value of the object.

What was Learned:

This assignment was a great learning experience for me as I did not have a lot of experience using Makefiles before this class. I had to do some extensive research on what each symbol ina Makefile stood for and how it needed to be formatted in order to function properly. This was really important as Makefiles would be a standard implementation feature in all the future programming assignments for the class and many of my other classes as well. This assignment also introduced the Boost test framework. This was completely new to me as well. I found it really interesting to work with and very useful as it could assure which functions were working correctly. The built in BOOST_REQUIRE functions were very easy to pick up on so jumping right in to creating my own boost tests was very simple.

PS1b

Linear Feedback Shift Register

PS1b: Makefile

```
1. CC = g++
2. CFLAGS = -c -g -Og -Wall -pedantic
3. OBJFILES = PhotoMagic.o FibLFSR.o
4. SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
5. TARGET = PhotoMagic
6.
7. all: $(TARGET)
8.
9. $(TARGET): $(OBJFILES)
10.             $(CC) $(OBJFILES) -o $(TARGET) $(SFMLFLAGS)
11.
12.     %.o: %.cpp $(DEPS)
13.             $(CC) $(CFLAGS) -o $@ $<
14.
15.     clean:
16.             rm -f $(OBJFILES) $(TARGET) *~
```

PS1b: PhotoMagic.cpp

```
1. #include <cstdlib>
2. #include <iostream>
3. #include <sstream>
4. #include <string>
5.
6. #include "FibLFSR.hpp"
7. #include <SFML/Graphics.hpp>
8. #include <SFML/Window.hpp>
9. #include <SFML/System.hpp>
10.
11.     using namespace std;
12.
13.     int main(int argc, char * argv[]) {
14.       string inputImage; //Argument 1 for main
15.       sf::Image input_image; //initialize SFML image for
    input image
16.       sf::Texture input_texture; //initialize SFML texture
    for input image
17.       sf::Sprite input_sprite; //initialize SFML sprite
    for input image
18.
19.       string outputImage; //Argument 2 for main
20.       sf::Image output_image; //initialize SFML image for
    output image
21.       sf::Texture output_texture; //initialize SFML
    texture for output image
22.       sf::Sprite output_sprite; //initialize SFML sprite
    for output image
23.
24.       string seed; //Argument 3 for main (is a seed for
    encoding or a tap for decoding)
25.       int tap;
26.
27.       sf::Color p; //p is a pixel
28.       sf::Vector2u size;
29.
30.       inputImage = argv[1]; //set arg1 to input image
31.       outputImage = argv[2]; //set arg2 to output image
32.       seed = argv[3]; //set arg3 to seed
33.       tap = atoi(argv[4]); //arg4 for tap
34.
35.       FibLFSR fiblfsr(seed, tap); //create LFSR object
36.
```

```
37.       if (!input_image.loadFromFile(inputImage)) { //check
  images
38.         cout << "Failed to load inputImage!" << endl;
39.         return -1;
40.       }
41.
42.       if (!output_image.loadFromFile(outputImage)) {
43.         cout << "Failed to load outputImage!" << endl;
44.         return -1;
45.       }
46.
47.       size = input_image.getSize();
48.       sf::RenderWindow inputWindow(sf::VideoMode(size.x*2,
  size.y), "INPUT IMAGE"); //create windows for encrypt and
  decrypt
49.       sf::RenderWindow
  outputWindow(sf::VideoMode(size.x*2, size.y), "OUTPUT
  IMAGE");
50.
51.       for (unsigned xAxis = 0; xAxis < size.x; xAxis++)
  { //for each pixel RGB tap to encrypt/decrypt
52.         for(unsigned yAxis = 0; yAxis < size.y; yAxis++) {
53.           p = input_image.getPixel(xAxis, yAxis);
54.           p.r = p.r ^ fiblfsr.generate(tap);
55.           p.g = p.g ^ fiblfsr.generate(tap);
56.           p.b = p.b ^ fiblfsr.generate(tap);
57.           output_image.setPixel(xAxis, yAxis, p);
58.         }
59.       }
60.
61.       input_texture.loadFromImage(input_image); //load
  textures and sprites
62.       input_sprite.setTexture(input_texture);
63.
64.       output_texture.loadFromImage(output_image);
65.       output_sprite.setTexture(output_texture);
66.
67.       while (inputWindow.isOpen() &&
  outputWindow.isOpen()) { //open windows and create event
  for display
68.         sf::Event event;
69.         while (inputWindow.pollEvent(event)) {
70.           if (event.type == sf::Event::Closed) {
71.          inputWindow.close();
72.           }
73.         }
74.         while (outputWindow.pollEvent(event)) {
```

```
75.           if (event.type == sf::Event::Closed) {
76.          outputWindow.close();
77.           }
78.         }
79.         inputWindow.clear(); //clear each window and
  display sprites
80.         inputWindow.draw(input_sprite);
81.         inputWindow.display();
82.         outputWindow.clear();
83.         outputWindow.draw(output_sprite);
84.         outputWindow.display();
85.       }
86.
87.       if (!output_image.saveToFile(outputImage)) { //save
  encrypted image to new file
88.         cout << "outputImage failed to save!" << endl;
89.         return -1;
90.       }
91.
92.       return 0;
93.     }
```

PS1b: FibLFSR.h

```
1. #ifndef LFSR_H
2. #define LFSR_H
3.
4. #include <string>
5. #include <iostream>
6.
7. class FibLFSR {
8. public:
9.     FibLFSR(std::string seed, int tap);
10.         FibLFSR(std::string seed);
11.         ~FibLFSR();
12.         int step();
13.         int generate(int k);
14.         friend std::ostream& operator<< (std::ostream
   &out, const FibLFSR& fiblfsr)\
15.     ;
16.     private:
17.         std::string Seed;//stores seed
18.         int rightMost;//stores rightmost
19.     };
20.
21.     #endif
```

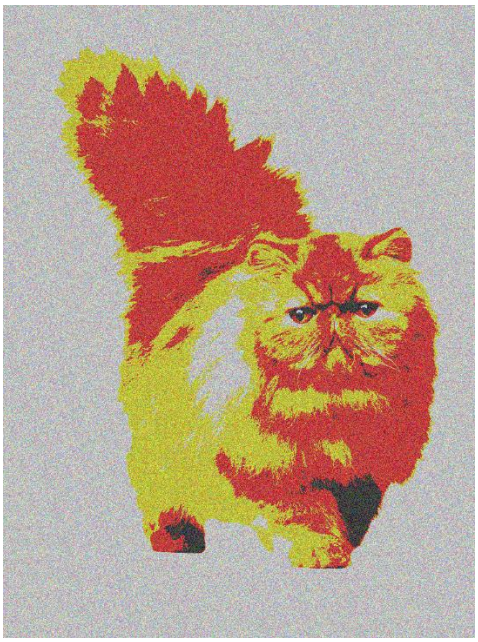
PS1b: FibLFSR.cpp

```
1. #include <iostream>
2. #include <string>
3. #include "FibLFSR.hpp"
4. #include <string>
5. #include <stdlib.h>
6. #include <math.h>
7.
8. using namespace std;
9. /*//main for quick testing
10.     int main() {
11.         FibLFSR l1 = FibLFSR("1011011000110110");
12.         cout << l1 << endl;
13.         l1.step();
14.         cout << l1 << endl;
15.         l1.step();
16.         cout << l1 << endl;
17.         l1.step();
18.         cout << l1 << endl;
19.         l1.step();
20.         cout << l1 << endl;
21.         l1.step();
22.         cout << l1 << endl;
23.         l1.step();
24.         cout << l1 << endl;
25.         l1.step();
26.         cout << l1 << endl;
27.         l1.step();
28.         cout << l1 << endl;
29.         l1.step();
30.         cout << l1 << endl;
31.         l1.step();
32.         cout << l1 << endl << endl;
33.
34.         cout << "TEST GENERATE" << endl;
35.         FibLFSR l2 = FibLFSR("1100011011000011");
36.         l2.generate(5);
37.         cout << l2 << endl;
38.         return 0;
39.     }
40.     */
41.     FibLFSR::FibLFSR(string seed, int tap) {
42.       Seed = seed;
43.       rightMost = tap;
```

```
44.     }
45.
46.     FibLFSR::FibLFSR(string seed) {
47.         Seed = seed;//initialize seed as input
48.         int a0 = Seed[0] - 48;//initialize rightmost
49.         int a2 = Seed[2] - 48;
50.         int a3 = Seed[3] - 48;
51.         int a5 = Seed[5] - 48;
52.         int xor1 = (a3 ^ a5);
53.         int xor2 = (a0 ^ a2);
54.         rightMost = xor1 ^ xor2;
55.     }
56.
57.     FibLFSR::~FibLFSR() {
58.         delete &Seed;
59.     }
60.
61.     ostream& operator<< (ostream& out, const FibLFSR&
   fiblfsr) {
62.         out << fiblfsr.Seed << " " << fiblfsr.rightMost <<
   endl; //print seed then rightmost
63.         return out;
64.     }
65.
66.     int FibLFSR::step() {
67.         int x0 = Seed[0] - 48;//cast chars from string to
   ints for xor
68.         int x2 = Seed[2] - 48;
69.         int x3 = Seed[3] - 48;
70.         int x5 = Seed[5] - 48;
71.         int xor1 = (x3 ^ x5);//xor by fibonacci pattern
72.         int xor2 = (x0 ^ x2);
73.         rightMost = xor1 ^ xor2;
74.         char xorChar = rightMost + 48;
75.         Seed.erase(0, 1);//erase first bit (shift)
76.         Seed = Seed + xorChar;//add next character to end
   of seed
77.         return rightMost;//update rightmost
78.
79.     }
80.
81.
82.     int FibLFSR::generate(int k) {
83.         int var1 = 0;
84.         for (int z = 0; z != k; z++) {
85.             int d = step();//call step function k times
```

```
            var1 += (d * pow(2, (z - 1)));//convert binary
  to decimal
        }
        rightMost = var1;  //update rightmost
        return rightMost; //return generated output
    }
```

PS1b: Output



PS1b: Discussion

What was Accomplished:

This assignment combined our knowledge of SFML and the LFSR to decode and encode an inputted image. The filename of a local image, the filename of the output image, and the Fibonacci LFSR seed was input and the pixels of the image were shifted via the LFSR. A main driver called PhotoMagic.cpp was created to run the SFML environment needed for this new functionality added on to PS1a. The main executable for the program would take in the input filename, output filename, and the seed string as mentioned above. From this, The input file would be read in and the generate function was ran on each pixel axis so that every pixel of the input image would be shifted and encoded. This program could also be used to decode the same image as long as the correct seed was input along with the encoded image. The output page shows the input image of a cat and the resulting encoded image below it.

Key Algorithms, Data Structures, OO Designs:

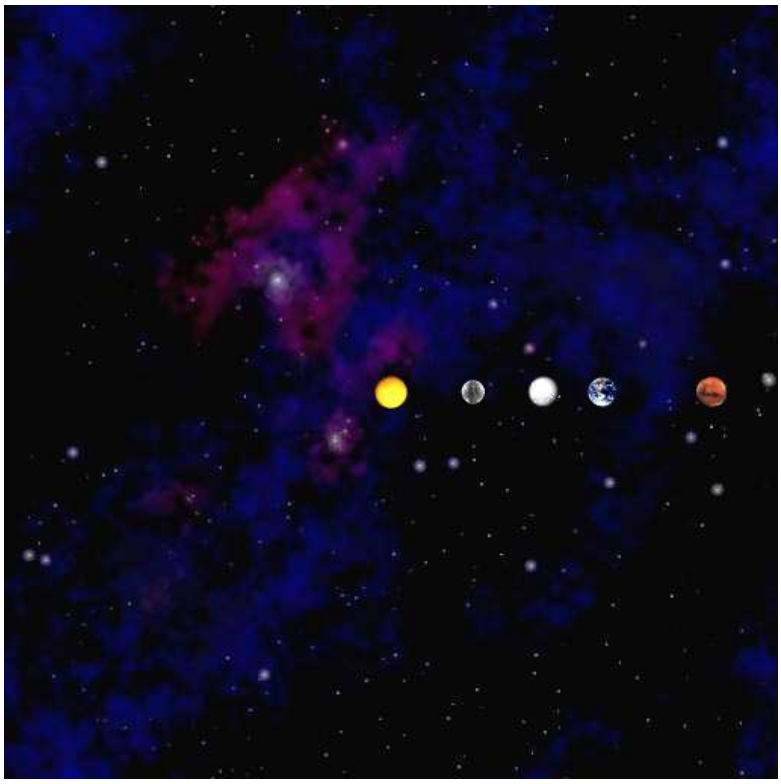
This assignment built off of the same principles exercised in the previous. Encapsulation and abstraction were used in the building of the FibLFSR class. Private variables stayed private where they needed to be and public functions only accessed them when they needed to. Polymorphism was still used in the case of overloading the stream insertion operator to display the current register of the FibLFSR object. SFML was used to input the image and output the encoded image to render windows. It was also necessary to access the pixels of the input image for the encoding process.

What was Learned:

This assignment really aided in fine tuning ideas that were explored in previous assignments. For instance, the SFML library was used in PS0, however in this assignment other functions such as getPixel and setPixel were used for the encoding of the input image. In this assignment, I also learned how to save an image to a new file for the encoded image had to be saved and differentiated from the inputted image. Another aspect I had learned from this assignment was how to check the number of arguments that came from the running the executable. In this case we needed four arguments (input image filename, output image filename, seed string, tap) and in order for the program to function correctly it was important to assign each argument to the correct variable as it was read in.

PS2a

N-Body Simulation



PS2a: Makefile

```
1. CC = g++
2. CFLAGS = -c -g -Og -Wall -pedantic -Werror
3. OBJFILES = main.o nbody.o
4. SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
5. TARGET = NBody
6.
7. all: $(TARGET)
8.
9. $(TARGET): $(OBJFILES)
10.             $(CC) $(OBJFILES) -o $(TARGET) $(SFMLFLAGS)
11.
12.     %.o: %.cpp $(DEPS)
13.             $(CC) $(CFLAGS) -o $@ $<
14.
15.     clean:
16.             rm -f $(OBJFILES) $(TARGET) *~
```

PS2a: main.cpp

```
1. #include "nbody.hpp"
2. #include <SFML/Graphics.hpp>
3. #include <SFML/Window.hpp>
4. #include <SFML/System.hpp>
5. #include <iostream>
6. #include <vector>
7. #include <sstream>
8. #include <cstdlib>
9. #include <memory>
10.
11.     using namespace std;
12.
13.     int main (int argc, char * argv[]) {
14.       string my_numOfPlanets; //holds file input for
  number of planets
15.       string my_universeRadius; //hold file input for
  radius of universe
16.       string input;
17.
18.       cin >> my_numOfPlanets; //input number of planets
19.       int numOfPlanets = atoi(my_numOfPlanets.c_str());
20.
21.       cin >> my_universeRadius; //input universe radius
22.       float universeRadius =
  atof(my_universeRadius.c_str());
23.
24.       Universe tempUniverse(numOfPlanets, universeRadius);
  //initialize universe
25.
26.       for (int n = 0; n < numOfPlanets; n++) { //create
  new celestial object for each input
27.         CelestialBody *temp = new CelestialBody();
  //create each object
28.         cin >> *temp; //take in input for each celestial
  body
29.         temp->setCoords(tempUniverse.getRadius()); //set
  the coordinates to each body
30.         tempUniverse.vectorPushBack(temp); //push to
  vector of pointers
31.       }
32.
33.       sf::RenderWindow nbodyWindow(sf::VideoMode(500,
  500), "NBody Universe"); //create render window for display
```

```
34.
35.       sf::Image backgroundImage; //create star filled
  background image from file
36.       if
   (!(backgroundImage.loadFromFile("starfield.jpg"))) {
37.         cout << "Cannot load background image!" << endl;
38.       }
39.       sf::Texture backgroundTexture;
40.       backgroundTexture.loadFromImage(backgroundImage);
41.       sf::Sprite backgroundSprite;
42.       backgroundSprite.setTexture(backgroundTexture);
43.
44.       while (nbodyWindow.isOpen()) { //open render window
  for display
45.         sf::Event event; //create event
46.         while(nbodyWindow.pollEvent(event)) {
47.           if (event.type == sf::Event::Closed) {
48.          nbodyWindow.close();
49.           }
50.         }
51.         nbodyWindow.clear(); //start with a clean slate
52.         nbodyWindow.draw(backgroundSprite); //add in
  background
53.         for (unsigned int k = 0; k !=
  tempUniverse.celestialVector.size(); k++) { //for each
  planet, draw
54.
  nbodyWindow.draw(*tempUniverse.celestialVector[k]);
55.         }
56.         nbodyWindow.display();
57.       }
58.       return 0;
59.     }
```

PS2a: nbody.h

```
1. #ifndef NBODY_HPP
2. #define NBODY_HPP
3.
4. #include <SFML/Graphics.hpp>
5. #include <SFML/Window.hpp>
6. #include <SFML/System.hpp>
7. #include <iostream>
8. #include <string>
9. #include <vector>
10.     #include <memory>
11.
12.     using namespace std;
13.
14.     class CelestialBody : public sf::Drawable { //defines
  each celestial body
15.     public:
16.       CelestialBody();
17.       CelestialBody(double my_xCoord, double my_yCoord,
  double my_xVelocity, double my_yVelocity, double my_mass,
  string my_planetName); //constucts via provided file inputs
18.       ~CelestialBody();
19.       void setCoords(float universeRadiuse);
20.       friend istream& operator>> (istream &input,
  CelestialBody &my_CelestialBody); //overloaded input
  operator
21.     private:
22.       virtual void draw(sf::RenderTarget& rendTarg,
  sf::RenderStates rendState) const;
23.       double xCoord; //x coordinate of planet
24.       double yCoord; //y coordinate of planet
25.       double xVelocity; //x velocity of planet
26.       double yVelocity; //y velocity of planet
27.       double mass; //mass of planet
28.       string planetName; //planet name
29.       sf::Image image;
30.       sf::Texture texture;
31.       sf::Sprite sprite;
32.     };
33.
34.     class Universe { //class universe will create the
  celestial objects
35.     public:
```

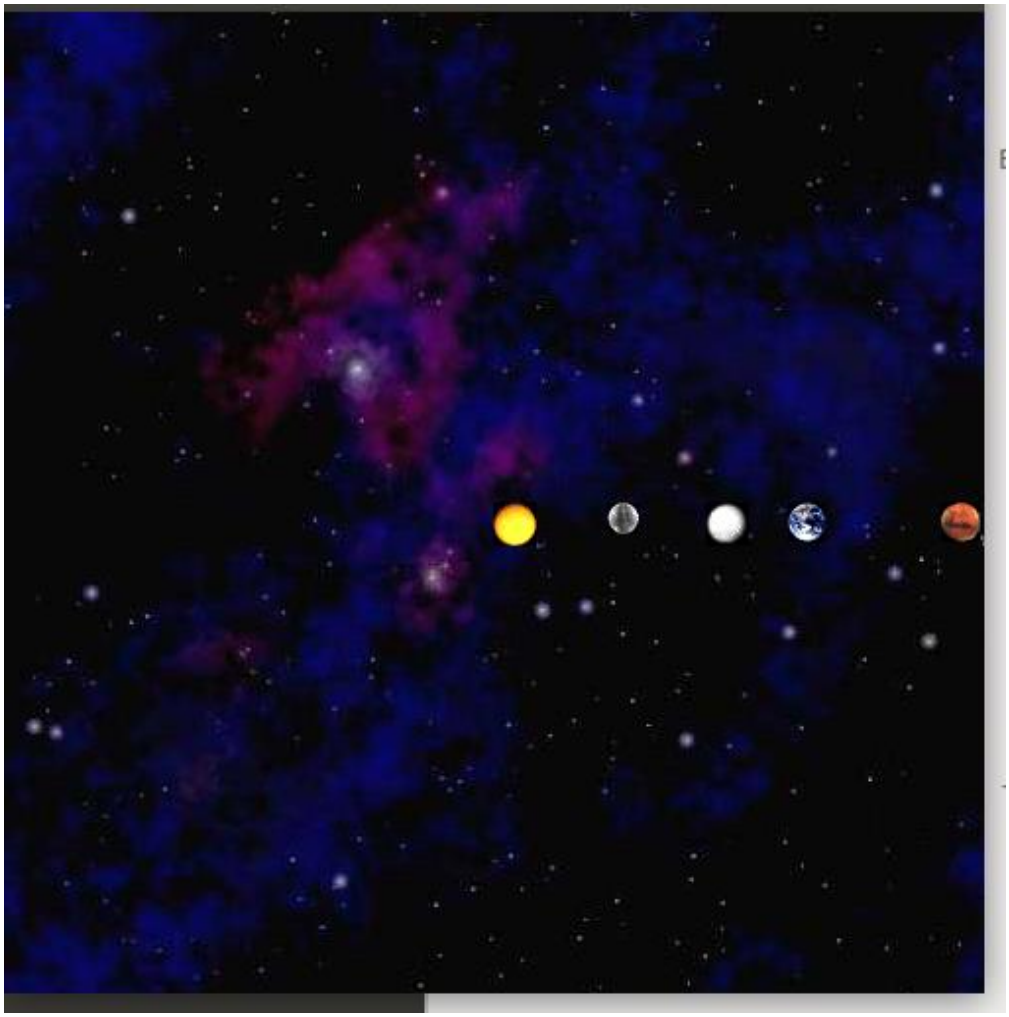
```
36.     Universe(int my_numOfPlanets, float
  my_universeRadius); //constucts universe
37.       void vectorPushBack(CelestialBody *CB);
38.       float getRadius(); //access radius
39.       vector<CelestialBody*> celestialVector; //vector of
  pointers for celestial bodies
40.     private:
41.       int numOfPlanets; //initializes number of planets
42.       float universeRadius; //initializes radius of
  universe
43.     };
44.
45.     #endif
```

PS2a: nbody.cpp

```
1. #include "nbody.hpp"
2. #include <iostream>
3. #include <string>
4. #include <memory>
5.
6. using namespace std;
7.
8. Universe::Universe(int my_numOfPlanets, float
   my_universeRadius) {
9.   numOfPlanets = my_numOfPlanets; //set the number of
   planets
10.       universeRadius = my_universeRadius; //set the
   universe radius
11.     }
12.
13.     void Universe::vectorPushBack(CelestialBody *CB) {
14.       this->celestialVector.push_back(CB);
15.     }
16.
17.     float Universe::getRadius() {
18.       return universeRadius;
19.     }
20.
21.     void CelestialBody::setCoords(float universeRadius) {
22.       float radius = universeRadius;
23.       xCoord = ((xCoord / radius) * (500 / 2)) + (500 /
   2);//set planet coords x
24.       yCoord = ((yCoord / radius) * (500 / 2)) + (500 /
   2);//set planet coords y
25.       sprite.setPosition(sf::Vector2f(xCoord, yCoord));
26.     }
27.
28.     CelestialBody::CelestialBody() { //initialize planet's
   properties as default
29.       xCoord = 0;
30.       yCoord = 0;
31.       xVelocity = 0;
32.       yVelocity = 0;
33.       mass = 0;
34.     }
35.
36.
```

```
37.     CelestialBody::CelestialBody(double my_xCoord, double
  my_yCoord, double my_xVelocity, double my_yVelocity, double
  my_mass, string my_planetName) { //initializes planet's
  properties by input
38.       xCoord = my_xCoord;
39.       yCoord = my_yCoord;
40.       xVelocity = my_xVelocity;
41.       yVelocity = my_yVelocity;
42.       mass = my_mass;
43.       texture.loadFromFile(my_planetName);
44.       sprite.setTexture(texture);
45.       sprite.setPosition(sf::Vector2f(xCoord, yCoord));
46.     }
47.
48.     CelestialBody::~CelestialBody() { //destructor
49.     }
50.
51.     void CelestialBody::draw(sf::RenderTarget& rendTarg,
  sf::RenderStates rendState)const { //draws celestial bodies
52.       rendTarg.draw(sprite);
53.     }
54.
55.     istream& operator>> (istream &input, CelestialBody
  &my_CelestialBody) { //takes inputs from file for celestial
  bodies
56.       input >> my_CelestialBody.xCoord;
57.       input >> my_CelestialBody.yCoord;
58.       input >> my_CelestialBody.xVelocity;
59.       input >> my_CelestialBody.yVelocity;
60.       input >> my_CelestialBody.mass;
61.       input >> my_CelestialBody.planetName;
62.
  my_CelestialBody.image.loadFromFile(my_CelestialBody.planet
  Name);
63.
  my_CelestialBody.texture.loadFromImage(my_CelestialBody.ima
  ge);
64.
  my_CelestialBody.sprite.setTexture(my_CelestialBody.texture
  );
65.
66.       return input;
67. }
```

PS2a: Output



PS2a: Discussion

What was Accomplished:

We returned again to SFML in assignment PS2a. This time an input text file was provided. The idea of this assignment was to read in images just as done with the previous assignment. The text file provided names of the planets in the solar system and each would be read in to populate an SFML window. Accompanying the planet image filenames was their x and r positions, their x and y velocities, and their masses. A Universe class was created to initialize the universe that would contain the planets. A CelestialBody class was created to initialize each planet as it was read in from the input text file. Each attribute mentioned from the input text file was a private member of the CelestialBody class so that each planet could be placed in the universe at the correct position to mimic the actual solar system's order. A Universe constructor created the universe, initializing its radius and the number of planets it would have. A push back function pushed each celestial body into the universe as it was read in from the input text file. The output page shows all the planets from the input text file displayed in their correct positions in the render window.

Key Algorithms, Data Structures, OO Designs:

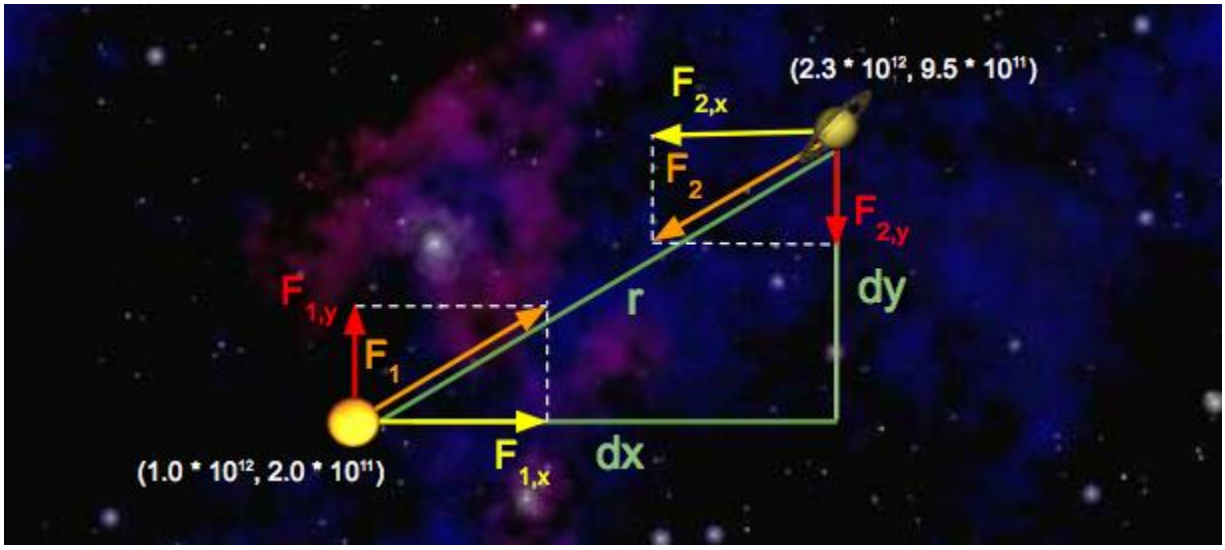
As always with classes, encapsulation and abstraction were practiced to make sure that private variables are private where they need to be and accessed by only the functions that need them. Polymorphism was used in the overloading of the input stream operator. In this occasion, the overloading provided the functionality of being able to load parameter data into an object. In the Universe class, a vector of pointers was used to store the CelestialBody objects as they were read in from the input text file. This allowed for easier access of the planets and the use of push back functions to populate the universe easier. Using a vector of pointers made managing the Universe a lot easier, if new planets needed to be added or existing planets needed to be removed it could be done easily in this case. Another key design feature this assignment included was the inheritance of the sf drawable class in the CelestialBody class. This allowed for the draw function to draw each sprite.

What was Learned:

This assignment helped reinforce knowledge of vector of pointers as it was required to implement this structure into the Universe class. This certainly helped in the CelestialBody management in the Universe. This assignment also expanded on the idea of reading arguments for the executable. This time it was a text file that contained the CelestialBody data within. More features of the SFML library were explored as well. SF RenderTargets and SF RenderStates were used in the draw function to draw the sprites.

PS2b

N-Body Simulation



PS2b: Makefile

```
1. CC = g++
2. CFLAGS = -c -g -Og -Wall -pedantic -Werror
3. OBJFILES = main.o nbody.o
4. SFMLFLAGS = -lsfml-audio -lsfml-graphics -lsfml-window -
   lsfml-system
5. TARGET = NBody
6.
7. all: $(TARGET)
8.
9. $(TARGET): $(OBJFILES)
10.         $(CC) $(OBJFILES) -o $(TARGET) $(SFMLFLAGS)
11.
12. %.o: %.cpp $(DEPS)
13.         $(CC) $(CFLAGS) -o $@ $<
14.
15. clean:
16. rm -f $(OBJFILES) $(TARGET) *~
```

PS2b: main.cpp

```
1. #include "nbody.hpp"
2. #include <SFML/Audio.hpp>
3. #include <SFML/Graphics.hpp>
4. #include <SFML/Window.hpp>
5. #include <SFML/System.hpp>
6. #include <iostream>
7. #include <vector>
8. #include <sstream>
9. #include <cstdlib>
10.     #include <memory>
11.     #include <cmath>
12.
13.     using namespace std;
14.
15.     int main (int argc, char * argv[]) {
16.       string my_numOfPlanets; //holds file input for
  number of planets
17.       string my_universeRadius; //hold file input for
  radius of universe
18.       string input;
19.       double rendTime;
20.       double stepTime;
21.       double timePassed = 0.0;
22.
23.       if (argc != 3) {
24.         cout << "Incorrect input!" << endl;
25.       }
26.
27.       rendTime = atof(argv[1]); //take in first argument
28.       stepTime = atof(argv[2]); //take in second argument
29.
30.       cin >> my_numOfPlanets; //input number of planets
31.       int numOfPlanets = atoi(my_numOfPlanets.c_str());
32.
33.       cin >> my_universeRadius; //input universe radius
34.       float universeRadius =
  atof(my_universeRadius.c_str());
35.
36.       Universe tempUniverse(numOfPlanets, universeRadius);
  //initialize universe
37.
38.       for (int n = 0; n < numOfPlanets; n++) { //create
  new celestial object for each input
```

```
39.         CelestialBody *temp = new CelestialBody();
  //create each object
40.         cin >> *temp; //take in input for each celestial
  body
41.         temp->setCoords(tempUniverse.getRadius()); //set
  the coordinates to each body
42.         tempUniverse.vectorPushBack(temp); //push to
  vector of pointers
43.       }
44.
45.       sf::Font arial; //add new font for timer
46.       if (!(arial.loadFromFile("arial.ttf"))) { //load ttf
  file for font
47.         cout << "Cannot load arial font!" << endl; //make
  sure it opens
48.       }
49.       sf::Text timer("0.0", arial); //create textfield for
  timer using font
50.
51.       sf::Music sound; //play music throughout simulation
52.       if (!(sound.openFromFile("2001.wav"))) { //check
  file opens
53.         cout << "Cannot load sound!" << endl;
54.       }
55.       sound.setVolume(10);
56.       sound.play();
57.
58.       sf::RenderWindow nbodyWindow(sf::VideoMode(500,
  500), "NBody Universe"); //create render window for display
59.
60.       sf::Image backgroundImage; //create star filled
  background image from file
61.       if
  (!(backgroundImage.loadFromFile("starfield.jpg"))) {
62.         cout << "Cannot load background image!" << endl;
63.       }
64.       sf::Texture backgroundTexture;
65.       backgroundTexture.loadFromImage(backgroundImage);
66.       sf::Sprite backgroundSprite;
67.       backgroundSprite.setTexture(backgroundTexture);
68.
69.       while (nbodyWindow.isOpen()) { //open render window
  for display
70.         sf::Event event; //create event
71.         while(nbodyWindow.pollEvent(event)) {
72.           if (event.type == sf::Event::Closed) {
73.          nbodyWindow.close();
```

```
74.           }
75.         }
76.         nbodyWindow.clear(); //start with a clean slate
77.
78.         int x; //incremental variable for each planet
79.         int y; //incremental variable to compare each
  planet to each planet
80.         x = 0;
81.         for (unsigned int l = 0; l !=
  tempUniverse.celestialVector.size(); l++) { //cycle through
  list of planets
82.           y = 0;
83.           double xforce = 0;
84.           double yforce = 0;
85.           for (unsigned int m = 0; m !=
  tempUniverse.celestialVector.size(); m++) { //cycle through
  each planet
86.          if (l != m) { //calculate the forces
87.            xforce = xforce +
  tempUniverse.celestialVector[l]->calcXForce(*tempUniverse.c
  elestialVector[x], *tempUniverse.celestialVector[y]);
88.            yforce = yforce +
  tempUniverse.celestialVector[l]->calcYForce(*tempUniverse.c
  elestialVector[x], *tempUniverse.celestialVector[y]);
89.          }
90.          y++;
91.           }
92.
  tempUniverse.celestialVector[l]->setXForce(xforce);
  //update force x
93.
  tempUniverse.celestialVector[l]->setYForce(yforce);
  //update force y
94.           x++;
95.         }
96.
97.         nbodyWindow.draw(backgroundSprite); //add in
  background
98.         for (unsigned int k = 0; k !=
  tempUniverse.celestialVector.size(); k++) { //for each
  planet, draw
99.
  tempUniverse.celestialVector[k]->setCoords(tempUniverse.get
  Radius());
100.
  nbodyWindow.draw(*tempUniverse.celestialVector[k]);
```

```
101.          tempUniverse.step(stepTime,
  tempUniverse.celestialVector[k]);
102.        }
103.
104.        timePassed = timePassed + stepTime; //check to see
  if program is finished
105.        if (timePassed > rendTime) {
106.          nbodyWindow.close(); //if so, close window
107.        }
108.        stringstream timerText; //for conversion of time
  passed to string for timer
109.        timerText << timePassed;
110.        timer.setString(timerText.str()); //display
  elapsed time
111.        nbodyWindow.draw(timer);
112.        nbodyWindow.display();
113.      }
114.      for (unsigned int h = 0; h !=
  tempUniverse.celestialVector.size(); h++) {
115.        cout << *tempUniverse.celestialVector[h] << endl;
116.      }
117.
118.      return 0;
119.    }
```

PS2b: nbody.h

```
1. #ifndef NBODY_HPP
2. #define NBODY_HPP
3.
4. #include <SFML/Audio.hpp>
5. #include <SFML/Graphics.hpp>
6. #include <SFML/Window.hpp>
7. #include <SFML/System.hpp>
8. #include <iostream>
9. #include <string>
10.     #include <vector>
11.     #include <memory>
12.
13.     const double gravity = 6.67e-11; //force of gravity
   constant
14.
15.     using namespace std;
16.
17.     class CelestialBody : public sf::Drawable { //defines
   each celestial body
18.     public:
19.       CelestialBody();
20.       CelestialBody(double my_xCoord, double my_yCoord,
   double my_xVelocity, double my_yVelocity, double my_mass,
   string my_planetName); //constucts via provided file inputs
21.       ~CelestialBody(); //getters and setters listed below
   (helpers)
22.       double getXVelocity();
23.       double getYVelocity();
24.       double getXCoord();
25.       double getYCoord();
26.       double getMass();
27.       double getXForce();
28.       double getYForce();
29.       double getXAcceleration();
30.       double getYAcceleration();
31.       string getPlanetName();
32.       void setXVelocity(double newXVelocity);
33.       void setYVelocity(double newYVelocity);
34.       void setXCoord(double newXCoord);
35.       void setYCoord(double newYCoord);
36.       void setMass(double newMass);
37.       void setCoords(float universeRadiuse);
38.       void setXForce(double newXForce);
```

```
39.       void setYForce(double newYForce);
40.       void setXAcceleration(double newXAcceleration);
41.       void setYAcceleration(double newYAcceleration);
42.       double calcXForce(CelestialBody &CB1, CelestialBody
  &CB2); //calculate forces on body
43.       double calcYForce(CelestialBody &CB1, CelestialBody
  &CB2);
44.       friend ostream& operator<< (ostream &output,
  CelestialBody &CB); //overloaded output operator
45.       friend istream& operator>> (istream &input,
  CelestialBody &my_CelestialBody); //overloaded input
  operator
46.     private:
47.       virtual void draw(sf::RenderTarget& rendTarg,
  sf::RenderStates rendState) const;
48.       double xAcceleration; //x acceleration
49.       double yAcceleration; //y acceleration
50.       double xForce; //x force
51.       double yForce; //y force
52.       double xCoord; //x coordinate of planet
53.       double yCoord; //y coordinate of planet
54.       double xVelocity; //x velocity of planet
55.       double yVelocity; //y velocity of planet
56.       double mass; //mass of planet
57.       string planetName; //planet name
58.       sf::Image image;
59.       sf::Texture texture;
60.       sf::Sprite sprite;
61.     };
62.
63.     class Universe { //class universe will create the
  celestial objects
64.     public:
65.       Universe(int my_numOfPlanets, float
  my_universeRadius); //constucts universe
66.       void vectorPushBack(CelestialBody *CB);
67.       float getRadius(); //access radius
68.       void step(double seconds, CelestialBody *CB);
69.       vector<CelestialBody*> celestialVector; //vector of
  pointers for celestial bodies
70.     private:
71.       int numOfPlanets; //initializes number of planets
72.       float universeRadius; //initializes radius of
  universe
73.     };
74.
75.     #endif
```

PS2b: nbody.cpp

```
1. #include "nbody.hpp"
2. #include <iostream>
3. #include <string>
4. #include <memory>
5. #include <cmath>
6.
7. using namespace std;
8.
9. Universe::Universe(int my_numOfPlanets, float
   my_universeRadius) {
10.       numOfPlanets = my_numOfPlanets; //set the number of
   planets
11.       universeRadius = my_universeRadius; //set the
   universe radius
12.     }
13.
14.     void Universe::vectorPushBack(CelestialBody *CB) {
15.       this->celestialVector.push_back(CB);
16.     }
17.
18.     void Universe::step(double seconds, CelestialBody *CB)
   { //step animation based on input seconds
19.       double xA = CB->getXAcceleration(); //movement
   variables for x and y
20.       double xV = CB->getXVelocity();
21.       double xC = CB->getXCoord();
22.       double xF = CB->getXForce();
23.       double yA = CB->getYAcceleration();
24.       double yV = CB->getYVelocity();
25.       double yC = CB->getYCoord();
26.       double yF = CB->getYForce();
27.       double m = CB->getMass();
28.
29.       CB->setXAcceleration(xF / m); //calculation for
   acceleration
30.       CB->setYAcceleration(yF / m);
31.       xA = CB->getXAcceleration();
32.       yA = CB->getYAcceleration();
33.
34.       CB->setXVelocity(xV + xA * seconds); //calculation
   for velocity
35.       CB->setYVelocity(yV + yA * seconds);
36.       xV = CB->getXVelocity();
```

```
37.       yV = CB->getYVelocity();
38.
39.       CB->setXCoord(xC + xV * seconds); //update coords
40.       CB->setYCoord(yC + yV * seconds);
41.     }
42.
43.     float Universe::getRadius() { //get radius of universe
44.       return universeRadius;
45.     }
46.
47.     void CelestialBody::setXAcceleration(double
  newXAcceleration) {
48.       xAcceleration = newXAcceleration;
49.     }
50.
51.     void CelestialBody::setYAcceleration(double
  newYAcceleration) {
52.       yAcceleration = newYAcceleration;
53.     }
54.
55.     void CelestialBody::setXForce(double newXForce)
  { //set new x force
56.       xForce = newXForce;
57.     }
58.
59.     void CelestialBody::setYForce(double newYForce)
  { //set new y force
60.       yForce = newYForce;
61.     }
62.
63.     double CelestialBody::calcXForce(CelestialBody &CB1,
  CelestialBody &CB2) { //calculate force x
64.       double xDistance; //holds delta of x distance
65.       double yDistance; //holds delta of y distance
66.       double sum; // holds sum of deltas
67.       double temp; //temp to hold sqrt
68.       double force; //force calc
69.       double xforce; // force of x calculation
70.
71.       xDistance = CB2.xCoord - CB1.xCoord;
72.       yDistance = CB2.yCoord - CB1.xCoord;
73.       sum = (xDistance * xDistance) + (yDistance *
  yDistance);
74.       temp = sqrt(sum);
75.       force = (gravity * CB1.mass * CB2.mass) / sum;
76.       xforce = force * (xDistance / temp);
77.       return xforce;
```

```
78.     }
79.
80.     double CelestialBody::calcYForce(CelestialBody &CB1,
  CelestialBody &CB2) { //calculate force y
81.       double xDistance; //holds delta of x distance
82.       double yDistance; //holds delta of y distance
83.       double sum; //holds sum of deltas
84.       double temp; //temp to hold sqrt
85.       double force; //force cals
86.       double yforce; //force of y calc
87.
88.       xDistance = CB2.xCoord - CB1.xCoord;
89.       yDistance = CB2.yCoord - CB1.xCoord;
90.       sum = (xDistance * xDistance) + (yDistance *
  yDistance);
91.       temp = sqrt(sum);
92.       force = (gravity * CB1.mass * CB2.mass) / sum;
93.       yforce = force * (yDistance / temp);
94.       return yforce;
95.     }
96.
97.     double CelestialBody::getXForce() { //getters and
  setter listed below
98.       return xForce;
99.     }
100.
101.    double CelestialBody::getYForce() {
102.      return yForce;
103.    }
104.
105.    double CelestialBody::getXAcceleration() {
106.      return xAcceleration;
107.    }
108.
109.    double CelestialBody::getYAcceleration() {
110.      return yAcceleration;
111.    }
112.
113.    void CelestialBody::setXCoord(double newXCoord)
  { //set a new x coord
114.      xCoord = newXCoord;
115.    }
116.
117.    void CelestialBody::setYCoord(double newYCoord)
  { //set a new y coord
118.      yCoord = newYCoord;
119.    }
```

```
120.
121.    double CelestialBody::getXVelocity() { //access x
  velocity
122.      return xVelocity;
123.    }
124.
125.    double CelestialBody::getYVelocity() { //access y
  velocity
126.      return yVelocity;
127.    }
128.
129.    double CelestialBody::getXCoord() { //access x coord
130.      return xCoord;
131.    }
132.
133.    double CelestialBody::getYCoord() { //access y coord
134.      return yCoord;
135.    }
136.
137.    double CelestialBody::getMass() { //access mass
138.      return mass;
139.    }
140.
141.    string CelestialBody::getPlanetName() {
142.      return planetName;
143.    }
144.
145.    void CelestialBody::setXVelocity(double newXVelocity)
  { //set new x velocity
146.      xVelocity = newXVelocity;
147.    }
148.
149.    void CelestialBody::setYVelocity(double newYVelocity)
  { //set new y velocity
150.      yVelocity = newYVelocity;
151.    }
152.
153.    void CelestialBody::setMass(double newMass) { //set
  new mass
154.      mass = newMass;
155.    }
156.
157.    void CelestialBody::setCoords(float universeRadius) {
158.      float radius = universeRadius;
159.      double xC = ((xCoord / radius) * (500 / 2)) + (500 /
  2);//set planet coords x
```

```
160.      double yC = ((yCoord / radius) * (500 / 2)) + (500 /
  2);//set planet coords y
161.      sprite.setPosition(sf::Vector2f(xC, yC));
162.    }
163.
164.    CelestialBody::CelestialBody() { //initialize planet's
  properties as default
165.      xForce = 0;
166.      yForce = 0;
167.      xAcceleration = 0;
168.      yAcceleration = 0;
169.      xCoord = 0;
170.      yCoord = 0;
171.      xVelocity = 0;
172.      yVelocity = 0;
173.      mass = 0;
174.    }
175.
176.
177.    CelestialBody::CelestialBody(double my_xCoord, double
  my_yCoord, double my_xVelocity, double my_yVelocity, double
  my_mass, string my_planetName) { //initializes planet's
  properties by input
178.      xForce = 0;
179.      yForce = 0;
180.      xAcceleration = 0;
181.      yAcceleration = 0;
182.      xCoord = my_xCoord;
183.      yCoord = my_yCoord;
184.      xVelocity = my_xVelocity;
185.      yVelocity = my_yVelocity;
186.      mass = my_mass;
187.      texture.loadFromFile(my_planetName);
188.      sprite.setTexture(texture);
189.      sprite.setPosition(sf::Vector2f(xCoord, yCoord));
190.    }
191.
192.    CelestialBody::~CelestialBody() { //destructor
193.    }
194.
195.    void CelestialBody::draw(sf::RenderTarget& rendTarg,
  sf::RenderStates rendState)const { //draws celestial bodies
196.      rendTarg.draw(sprite);
197.    }
198.
199.    ostream& operator<< (ostream &output, CelestialBody
  &CB) { //output each variable of a planet object
```

```
200.      //output << "PLANET NAME: " << CB.getPlanetName();
201.      output << "PLANET NAME: " << CB.planetName << endl;
202.      output << "MASS: " << CB.getMass() << endl;
203.      output << "xCoord: " << CB.getXCoord() << endl;
204.      output << "yCoord: " << CB.getYCoord() << endl;
205.      output << "xVelocity: " << CB.getXVelocity() <<
  endl;
206.      output << "yVelocity: " << CB.getYVelocity() <<
  endl;
207.      output << "xForce: " << CB.getXForce() << endl;
208.      output << "yForce: " << CB.getYForce() << endl;
209.      output << "xAcceleration: " << CB.getXAcceleration()
  << endl;
210.      output << "yAcceleration: " << CB.getYAcceleration()
  << endl;
211.      return output;
212.    }
213.
214.
215.    istream& operator>> (istream &input, CelestialBody
  &my_CelestialBody) { //takes inputs from file for celestial
  bodies
216.      input >> my_CelestialBody.xCoord;
217.      input >> my_CelestialBody.yCoord;
218.      input >> my_CelestialBody.xVelocity;
219.      input >> my_CelestialBody.yVelocity;
220.      input >> my_CelestialBody.mass;
221.      input >> my_CelestialBody.planetName;
222.
  my_CelestialBody.image.loadFromFile(my_CelestialBody.planet
  Name);
223.
  my_CelestialBody.texture.loadFromImage(my_CelestialBody.ima
  ge);
224.
  my_CelestialBody.sprite.setTexture(my_CelestialBody.texture
  );
225.
226.      return input;
227.    }
```

PS2b: Output

```
PLANET NAME: earth.gif
MASS: 5.974e+24
xCoord: -2.32427e+14
yCoord: -2.31621e+14
xVelocity: -7.10788e+06
yVelocity: -7.08718e+06
xForce: 5.19486e+15
yForce: 5.19486e+15
xAcceleration: 8.69578e-10
yAcceleration: 8.69578e-10

PLANET NAME: mars.gif
MASS: 6.419e+23
xCoord: -7.02097e+12
yCoord: -6.23587e+12
xVelocity: -287071
yVelocity: -263012
xForce: 6.12048e+17
yForce: 6.12056e+17
xAcceleration: 9.53494e-07
yAcceleration: 9.53506e-07

PLANET NAME: mercury.gif
MASS: 3.302e+23
xCoord: -1.66016e+14
yCoord: -1.64091e+14
```

```
xVelocity: -4.16864e+06
yVelocity: -4.12157e+06
xForce: 5.62645e+14
yForce: 5.62645e+14
xAcceleration: 1.70395e-09
yAcceleration: 1.70395e-09

PLANET NAME: sun.gif
MASS: 1.989e+30
xCoord: 8.71535e+06
yCoord: 9.42482e+06
xVelocity: 0.216936
yVelocity: 0.263942
xForce: -2.51421e+19
yForce: -1.98745e+19
xAcceleration: -1.26406e-11
yAcceleration: -9.99219e-12

PLANET NAME: venus.gif
MASS: 4.869e+24
xCoord: -5.30299e+12
yCoord: -3.93985e+12
xVelocity: -145642
yVelocity: -110678
xForce: 8.13232e+18
yForce: 8.13244e+18
xAcceleration: 1.67022e-06
yAcceleration: 1.67025e-06
```

PLANET NAME: uranus.gif

MASS: 3.714e+24

xCoord: -4.23924e+12

yCoord: -3.51844e+12

xVelocity: -156626

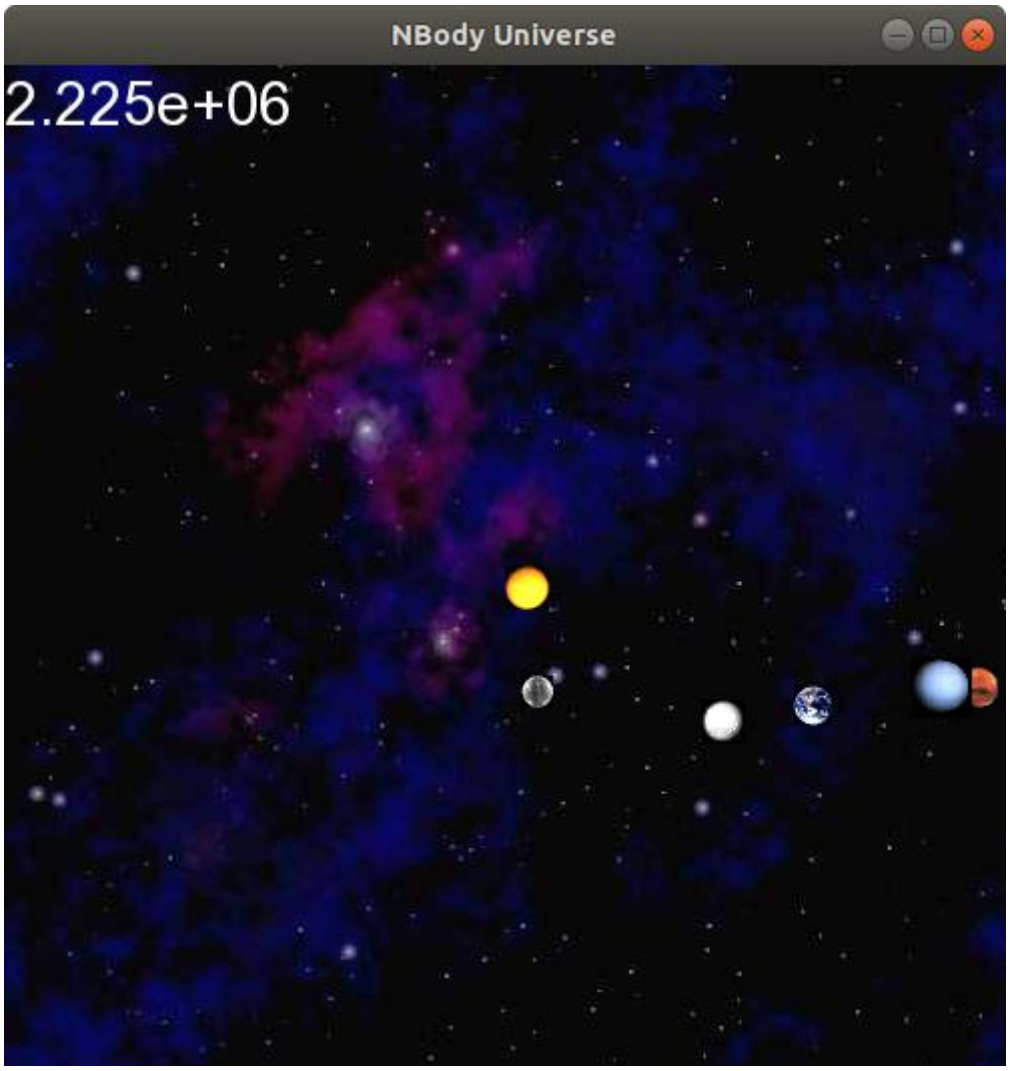
yVelocity: -134628

xForce: 9.71044e+18

yForce: 9.71166e+18

xAcceleration: 2.61455e-06

yAcceleration: 2.61488e-06



PS2b: Discussion

What was Accomplished:

This assignment expanded on the previous as physics was assigned to the planets using the data that was assigned to them from the input text file as they were loaded into the Universe. Mutators were first added in to the CelestialBody class so that the velocities of each celestial body could be altered by the simulation. A step function was then added in to step the planets into their next position each new frame. This would allow for the movement of the planets. This function took in a time parameter so the planets would only move for a set amount of time. The positions of each planet still needed to be calculated and updated every frame in order for motion to occur. This was done through calculating the net force and acceleration in pairwise force for each planet. Using the provided Newtonian equations, the planets' positions could be updated. An issue that I ran into during this assignment was The planets would get launched out of view of the render window. I believe this issue may have originated in my calculate x and y force functions or my step function of my CelestialBody class. What seemed to be happening was a the positions for the planets was being skewed somehow. This could have been an incorrect calculation of the forced or an error in setting and calculating the velocity and accelerations in the step function. Both could have influenced the result for position. For extra credit, the elapsed time was displayed in the render window, sound was played and a new universe was created as well (a universe including Uranus). The output page shows the attributes of each planet. The image after shows the time and that the planets are in different positions than where they started at. All the planets begin inline as they did in the previous assignment so this shows that they are in motion. This image also shows the new universe I created as it has Uranus in it as well.

Key Algorithms, Data Structures, OO Designs:

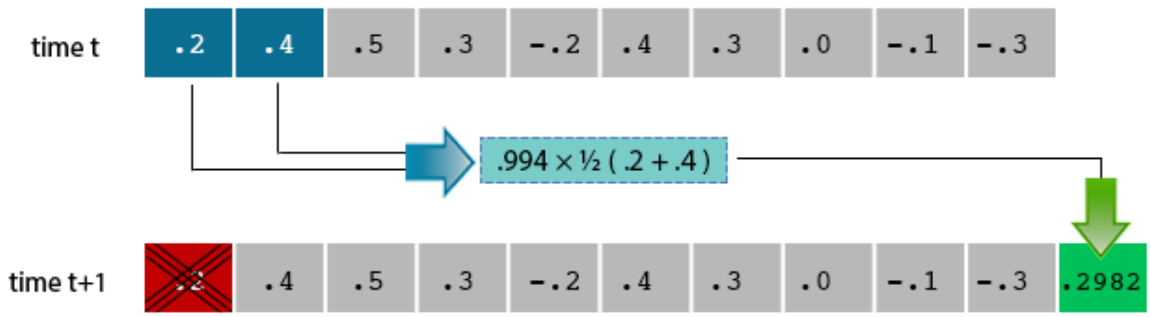
Similar to the previous part of this assignment, encapsulation and abstraction were exercised on the classes that were made. Accessor and mutator functions were used in the accessing and altering of the CelestialBody class private member variables. The Celestial Body still inherited the SF drawable class as well for the functionality of rendering the sprites in the draw function. Polymorphism was still present with the overloading of both the output and input operators. The input operator allowed for the creation and initialization of each planet while the ouput operator showed the status of each planet. The vector of pointers made it easy to create a new universe as all that was required was for a new input text file to be created with an additions to be made added in, in my case it was the planet Uranus.

What was Learned:

This assignment really expanded on the animating abilities of the SFML API. Using the mathematical equations for calculating net forces and acceleration, the planets could be animated into orbit (if done correctly). I also discovered how to add music to the render window animation using SF Music. Each assignment really expanded on a new area of functionality of the SFML library. Using the draw functions the planets could be continually re rendered to update to their new positions and create animation. I also learned how to use the timer to control how long an animation would last. There was a lot of SFML functionality that was discovered in this particular assignment.

PS3a

Synthesizing a Plucked String Sound



the Karplus-Strong update

PS3a: Makefile

```
1. CC = g++
2. CFLAGS = -std=c++17 -Wall -pedantic -Werror
3. #SFMLFLAGS = -lsfml-audio -lsfml-graphics -lsfml-window -
   lsfml-system
4. Boost = -lboost_unit_test_framework
5.
6. all: GuitarString
7.
8. GuitarString: CircularBuffer.o test.o
9.         $(CC) -o GuitarString CircularBuffer.o test.o
   $(Boost)
10.
11.     test.o: test.cpp CircularBuffer.h
12.             $(CC) -c test.cpp -o test.o $(CFLAGS)
13.
14.     CircularBuffer.o: CircularBuffer.cpp CircularBuffer.h
15.             $(CC) -c CircularBuffer.cpp -o
   CircularBuffer.o $(CFLAGS)
16.
17.     clean:
18.     rm -rf GuitarString *.o *.~
```

PS3a: test.cpp

```
1. // Copyright [2020] Jacob
2.
3. #define BOOST_TEST_DYN_LINK
4. #define BOOST_TEST_MODULE Main
5.
6. #include <stdint.h>
7. #include <iostream>
8. #include <string>
9. #include <exception>
10.     #include <stdexcept>
11.     #include "CircularBuffer.h"
12.     #include <boost/test/unit_test.hpp>
13.
14.     BOOST_AUTO_TEST_CASE(CircularBuffer_CONSTRUCTOR) {
15.       BOOST_REQUIRE_NO_THROW(CircularBuffer(10));  //
   check constructor
16.       BOOST_REQUIRE_THROW(CircularBuffer(0),
   std::exception);
17.       BOOST_REQUIRE_THROW(CircularBuffer(0),
   std::invalid_argument);
18.     }
19.
20.     BOOST_AUTO_TEST_CASE(CircularBuffer_ISEMPTY) {
21.       CircularBuffer circbuff(2);  // add and remove then
   check if empty
22.       circbuff.enqueue(1);
23.       circbuff.enqueue(2);
24.       circbuff.dequeue();
25.       circbuff.dequeue();
26.       BOOST_REQUIRE(circbuff.isEmpty() == true);
27.     }
28.
29.     BOOST_AUTO_TEST_CASE(CircularBuffer_ISFULL) {
30.       CircularBuffer circbuff(2);  // add in elements to
   capacity and check if full
31.       circbuff.enqueue(1);
32.       circbuff.enqueue(2);
33.       BOOST_REQUIRE(circbuff.isFull() == true);
34.     }
35.
36.     BOOST_AUTO_TEST_CASE(CircularBuffer_ENQUEUE) {
37.       CircularBuffer circbuff(2);  // add elements and
   check enqueue works
```

```
38.       circbuff.enqueue(1);
39.       circbuff.enqueue(2);
40.       BOOST_REQUIRE_THROW(circbuff.enqueue(3),
  std::runtime_error);
41.     }
42.
43.     BOOST_AUTO_TEST_CASE(CircularBuffer_DEQUEUE) {
44.       CircularBuffer circbuff(2);  // try to dequeue when
  empty
45.       circbuff.enqueue(1);
46.       circbuff.enqueue(2);
47.       circbuff.dequeue();
48.       circbuff.dequeue();
49.       BOOST_REQUIRE_THROW(circbuff.dequeue(),
  std::runtime_error);
50.     }
51.
52.     BOOST_AUTO_TEST_CASE(CircularBuffer_PEEK) {
53.       CircularBuffer circbuff(2);  // check peek works
54.       circbuff.enqueue(1);
55.       circbuff.enqueue(2);
56.       BOOST_REQUIRE(circbuff.peek() == 1);
57.       circbuff.dequeue();  // check peek does not work
  when empty
58.       circbuff.dequeue();
59.       BOOST_REQUIRE_THROW(circbuff.peek(),
  std::runtime_error);
60.     }
61.
62.     BOOST_AUTO_TEST_CASE(CircularBuffer_SIZE) {
63.       CircularBuffer circbuff(2);  // check size works
64.       circbuff.enqueue(1);
65.       circbuff.enqueue(2);
66.       BOOST_REQUIRE(circbuff.size() == 2);
67.     }
```

PS3a: CircularBuffer.h

```
1. // Copyright [2020] Jacob
2.
3. #ifndef
   _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS3A_CIRCULARBUFFER_H_
4. #define
   _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS3A_CIRCULARBUFFER_H_
5.
6. #include <stdint.h>
7. #include <iostream>
8. #include <SFML/Audio.hpp>
9. #include <SFML/Graphics.hpp>
10.     #include <SFML/Window.hpp>
11.     #include <SFML/System.hpp>
12.
13.     class CircularBuffer {
14.      public:
15.       explicit CircularBuffer(int init_capacity);
16.       ~CircularBuffer();
17.       int size();
18.       bool isEmpty();
19.       bool isFull();
20.       void enqueue(int16_t newData);
21.       int16_t dequeue();
22.       int16_t peek();
23.      private:
24.       int capacity;
25.       int arrSize;
26.       int16_t front;
27.       int16_t back;
28.       int16_t* arr;
29.     };
30.
31.     #endif  //
   _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS3A_CIRCULARBUFFER_H_
```

PS3a: CircularBuffer.cpp

```
1. // Copyright [2020] Jacob
2.
3. #include "CircularBuffer.h"
4. #include <stdint.h>
5. #include <stdexcept>
6. #include <exception>
7. #include <iostream>
8. #include <string>
9. #include <SFML/Audio.hpp>
10.     #include <SFML/Graphics.hpp>
11.     #include <SFML/Window.hpp>
12.     #include <SFML/System.hpp>
13.
14.
15.     CircularBuffer::CircularBuffer(int init_capacity) {
16.       if (init_capacity <= 1) {  // check capacity input
  or throw error
17.         throw std::invalid_argument("CircularBuffer
  constructor: ");
18.         std::cout << "capacity must be greater than 0" <<
  std::endl;
19.       } else {  // initialize queue as array
20.         capacity = init_capacity;
21.         arrSize = 0;
22.         front = 0;
23.         back = 0;
24.         arr = new int16_t[capacity];
25.       }
26.     }
27.
28.     CircularBuffer::~CircularBuffer() {
29.     }
30.
31.     int CircularBuffer::size() {
32.       return arrSize;  // get circbuff size
33.     }
34.
35.     bool CircularBuffer::isEmpty() {
36.       if (arrSize == 0) {  // check if empty
37.         return true;
38.       } else {
39.         return false;
40.       }
```

```
41.     }
42.
43.     bool CircularBuffer::isFull() {
44.       if (arrSize == capacity) {  // check if full
45.         return true;
46.       } else {
47.         return false;
48.       }
49.     }
50.
51.     void CircularBuffer::enqueue(int16_t newData) {
52.       if (isFull() == true) {  // check if full
53.         throw std::runtime_error("enqueue: ");
54.         std::cout << "can't enqueue to a full ring" <<
  std::endl;
55.       } else {
56.         int index = (front + arrSize) % capacity;  //
  enqueue to correct position
57.         arr[index] = newData;  // add data to queue
58.         arrSize++;  // increase size
59.       }
60.     }
61.
62.     int16_t CircularBuffer::dequeue() {
63.       int temp = front;
64.       if (isEmpty() == true) {  // check if empty
65.         throw std::runtime_error("dequeue: ");
66.         std::cout << "can't dequeue an empty ring" <<
  std::endl;
67.       } else {
68.         arrSize--;  // decrement size
69.         front = (front + 1) % capacity;  // set new front
70.         return arr[temp];
71.       }
72.     }
73.
74.     int16_t CircularBuffer::peek() {
75.       if (isEmpty() == true) {  // check if empty
76.         throw std::runtime_error("peek: ");
77.         std::cout << "can't peek an empty ring" <<
  std::endl;
78.       } else {
79.         return arr[front];  // peek front
80.       }
81.     }
```

PS3a: Output

```
osboxes@osboxes:~/Documents$ cd COMP\ IV\ PS3a
osboxes@osboxes:~/Documents/COMP IV PS3a$ ls
 CircularBuffer.cpp     CircularBuffer.o   PS3a_JacobBlumsack          test.cpp~
 CircularBuffer.cpp~    GuitarString       PS3a_JacobBlumsack.tar.xz   test.o
 CircularBuffer.h       Makefile           Ps3a-readme.odt
 CircularBuffer.h~     '#Makefile#'        Ps3a-readme.txt
 CircularBuffer.h.gch   Makefile~          test.cpp
osboxes@osboxes:~/Documents/COMP IV PS3a$ emacs Makefile
osboxes@osboxes:~/Documents/COMP IV PS3a$ ./GuitarString
Running 7 test cases...

*** No errors detected
osboxes@osboxes:~/Documents/COMP IV PS3a$ emacs CircularBuffer.cpp
osboxes@osboxes:~/Documents/COMP IV PS3a$ emacs test.cpp
osboxes@osboxes:~/Documents/COMP IV PS3a$
```

PS3a: Discussion

What was Accomplished:

In this assignment, a circular buffer was implemented and tested using the Boost test network. CPPLint was also used to make sure the program fell into certain style guidelines. This was the first part of the synthesizing a plucked string assignment so the extent of it only extended to implementing the circular buffer and testing that its functionality is correct. The circular buffer constructor creates an empty ring buffer with a given maximum. The buffer works via queue principles so two functions, enqueue and dequeue, were written to allow for items to be added and removed from the buffer. The test.cpp file uses boost test network to test each function implemented in the CircularBuffer class. The output page shows proof the assignment functioned properly. A boost test was constructed for each function of the class and the provided screen capture shows that there the boost tests all run with no errors.

Key Algorithms, Data Structures, OO Designs:

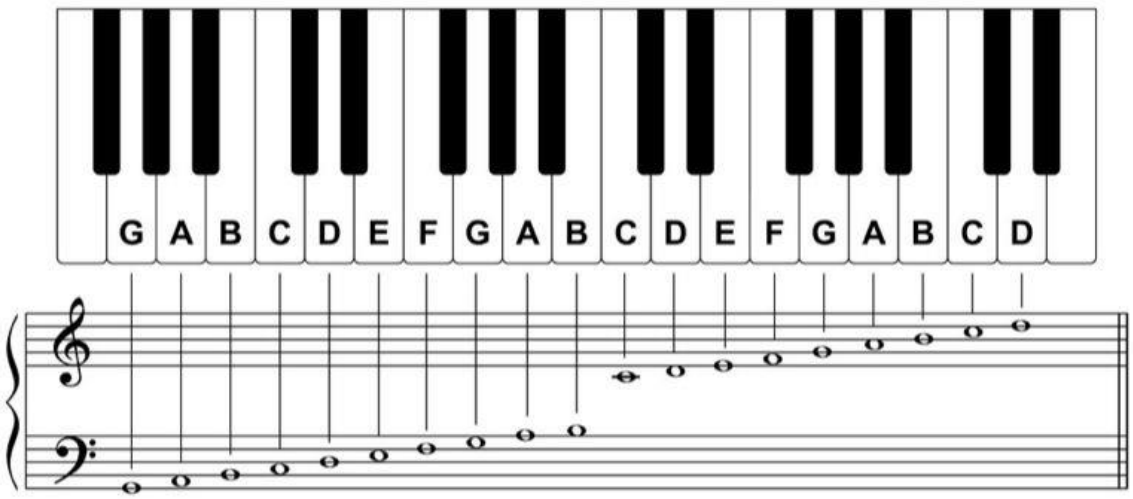
Being a program with a class, encapsulation and abstraction principles were applied. Private variables were made to track different attributes of the buffer such as the front, back, capacity, etc. The circular buffer was designed to function off queue based principles meaning it is first in first out (FIFO). To do this, an array was used and functions were designed to give it queue attributes such as enqueue and dequeue. The array worked very well as the underlying structure for the circular buffer as it allowed for each element to be easily accessed and indexed for queue implementation.

What was Learned:

This assignment introduced the CPPLint style guidelines. Once downloaded I found that it was easy to run the CPPLint style guidelines on each of my code files. I also found it easy to understand what needed to be adjusted in each file to comply with the style guidelines. There were some requirements that I felt were nit picky such as the line length requirement however for the most part CPPLint did a nice job holding my code to a consistent style. I had also never used the variable type int16_t. I now know that it is a signed integer with a width of exactly 16 bits without padding and that it uses 2's compliment for negative values. I also began using BOOST_REQUIRE_THROW which was another piece of functionality from the boost test network that allowed for the throwing of an error for functions when needed. For example in my code if I tried to call dequeue when the circular buffer was empty I would throw a runtime error. This assignment also used exception handling in the functions alongside the boost test network to make sure they were functioning correctly.

PS3b

Synthesizing a Plucked String Sound



PS3b: Makefile

```
1. CC = g++
2. CFLAGS = -std=c++17 -Wall -pedantic -Werror
3. SFMLFLAGS = -lsfml-audio -lsfml-graphics -lsfml-window -
   lsfml-system
4. Boost = -lboost_unit_test_framework
5.
6. all: KSGuitarSim
7.
8. KSGuitarSim: KSGuitarSim.o StringSound.o CircularBuffer.o
9.         $(CC) $(CFLAGS) -o KSGuitarSim KSGuitarSim.o
   StringSound.o CircularBuffer.o $(SFMLFLAGS)
10.
11.     test.o: test.cpp CircularBuffer.h
12.             $(CC) $(CFLAGS) -c test.cpp -o test.o
   $(CFLAGS)
13.
14.     KSGuitarSim.o: KSGuitarSim.cpp CircularBuffer.h
   StringSound.h
15.             $(CC) $(CFLAGS) -c KSGuitarSim.cpp -o
   KSGuitarSim.o $(SFMLFLAGS)
16.
17.     StringSound.o: StringSound.cpp StringSound.h
   CircularBuffer.h
18.             $(CC) $(CFLAGS) -c StringSound.cpp -o
   StringSound.o $(SFMLFLAGS)
19.
20.     CircularBuffer.o: CircularBuffer.cpp CircularBuffer.h
21.             $(CC) $(CFLAGS) -c CircularBuffer.cpp -o
   CircularBuffer.o $(CFLAGS)
22.
23.     clean:
24.             rm -rf KSGuitarSim *.o *.~
```

PS3b: KSGuitarSim.cpp

```
1. #include <SFML/Graphics.hpp>
2. #include <SFML/System.hpp>
3. #include <SFML/Audio.hpp>
4. #include <SFML/Window.hpp>
5.
6. #include <math.h>
7. #include <limits.h>
8.
9. #include <iostream>
10.     #include <string>
11.     #include <exception>
12.     #include <stdexcept>
13.     #include <vector>
14.
15.     #include "CircularBuffer.h"
16.     #include "StringSound.h"
17.
18.     #define CONCERT_A 220.0
19.     #define SAMPLES_PER_SEC 44100
20.
21.     std::vector<sf::Int16> makeSamples(StringSound ss)
  { //makesample function
22.       std::vector<sf::Int16> samples; //vector for samples
23.
24.       ss.pluck();
25.       int duration = 8;  // seconds
26.       int i;
27.       for (i= 0; i < SAMPLES_PER_SEC * duration; i++)
  { //calculate duration for tic
28.         ss.tic();
29.         samples.push_back(ss.sample());
30.       }
31.
32.       return samples;
33.     }
34.
35.     int main() {
36.       std::string keyboardInput = "q2we4r5ty7u8i9op-
  [=zxdcfvgbnjmk,.;/' "; //piano style string for keyboard
  input
37.
38.       sf::RenderWindow window(sf::VideoMode(300, 200),
  "SFML Plucked String Sound Lite"); //open window
```

```
39.       double freq;
40.
41.       std::vector<std::vector<sf::Int16>> audioSample(37);
  //vector of vectors for audio sample
42.       std::vector<sf::SoundBuffer> soundBuffer(37);
  //vector for sound buffer
43.       std::vector<sf::Sound> sound(37); //vector for sound
44.
45.       for (int i = 0; i < 37; i++) { //for each keyboard
  input
46.         freq = CONCERT_A * pow(2, ((i / 24.0)/12.0));
47.         StringSound ss = StringSound(freq); //create tone
48.         std::vector<sf::Int16> samples; //vector for
  samples
49.         samples = makeSamples(ss);
50.         audioSample[i] = samples; //add sound to sound
  buffer vector
51.         if
  (!(soundBuffer[i]).loadFromSamples(&(audioSample[i][0]),
  audioSample[i].size(), 2, SAMPLES_PER_SEC)) {
52.          throw std::runtime_error("Failed Sound Buffer!");
53.           }
54.         sound[i].setBuffer(soundBuffer[i]);
55.       }
56.       while (window.isOpen()) { //while window is open
57.         sf::Event event; //initialize event
58.         while (window.pollEvent(event)) { //start event
59.           switch (event.type) {
60.           case sf::Event::Closed: //for closing of window
61.          window.close();
62.          break;
63.
64.           case sf::Event::TextEntered: //case for keyboard
  input
65.          if (event.text.unicode < 128) {
66.            char note =
  static_cast<char>(event.text.unicode); //take each note for
  each corresponding keyboard input
67.            for (unsigned int j = 0; j <
  keyboardInput.size(); j++) {
68.              if (keyboardInput[j] == note) { //match input
  with note
69.                std::cout << "Current note is at: " <<
  keyboardInput[j] << std::endl; //notify use of note
70.                sound[j].play(); //play the sound
71.                break;
72.              }
```

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

PS3b: CircularBuffer.h

```
1. // Copyright [2020] Jacob
2.
3. #ifndef
   _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS3A_CIRCULARBUFFER_H_
4. #define
   _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS3A_CIRCULARBUFFER_H_
5.
6. #include <stdint.h>
7. #include <iostream>
8. #include <SFML/Audio.hpp>
9. #include <SFML/Graphics.hpp>
10.     #include <SFML/Window.hpp>
11.     #include <SFML/System.hpp>
12.
13.     class CircularBuffer {
14.      public:
15.       explicit CircularBuffer(int init_capacity);
16.       ~CircularBuffer();
17.       int size();
18.       bool isEmpty();
19.       bool isFull();
20.       void enqueue(int16_t newData);
21.       int16_t dequeue();
22.       int16_t peek();
23.      private:
24.       int capacity;
25.       int arrSize;
26.       int16_t front;
27.       int16_t back;
28.       int16_t* arr;
29.     };
30.
31.     #endif  //
   _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS3A_CIRCULARBUFFER_H_
```

PS3b: CircularBuffer.cpp

```
1. // Copyright [2020] Jacob
2.
3. #include "CircularBuffer.h"
4. #include <stdint.h>
5. #include <stdexcept>
6. #include <exception>
7. #include <iostream>
8. #include <string>
9. #include <SFML/Audio.hpp>
10.     #include <SFML/Graphics.hpp>
11.     #include <SFML/Window.hpp>
12.     #include <SFML/System.hpp>
13.
14.
15.     CircularBuffer::CircularBuffer(int init_capacity) {
16.       if (init_capacity <= 1) {  // check capacity input
  or throw error
17.         throw std::invalid_argument("CircularBuffer
  constructor: ");
18.         std::cout << "capacity must be greater than 0" <<
  std::endl;
19.       } else {  // initialize queue as array
20.         capacity = init_capacity;
21.         arrSize = 0;
22.         front = 0;
23.         back = 0;
24.         arr = new int16_t[capacity];
25.       }
26.     }
27.
28.     CircularBuffer::~CircularBuffer() {
29.     }
30.
31.     int CircularBuffer::size() {
32.       return arrSize;  // get circbuff size
33.     }
34.
35.     bool CircularBuffer::isEmpty() {
36.       if (arrSize == 0) {  // check if empty
37.         return true;
38.       } else {
39.         return false;
40.       }
```

```
41.     }
42.
43.     bool CircularBuffer::isFull() {
44.       if (arrSize == capacity) {  // check if full
45.         return true;
46.       } else {
47.         return false;
48.       }
49.     }
50.
51.     void CircularBuffer::enqueue(int16_t newData) {
52.       if (isFull() == true) {  // check if full
53.         throw std::runtime_error("enqueue: ");
54.         std::cout << "can't enqueue to a full ring" <<
  std::endl;
55.       } else {
56.         int index = (front + arrSize) % capacity;  //
  enqueue to correct position
57.         arr[index] = newData;  // add data to queue
58.         arrSize++;  // increase size
59.       }
60.     }
61.
62.     int16_t CircularBuffer::dequeue() {
63.       int temp = front;
64.       if (isEmpty() == true) {  // check if empty
65.         throw std::runtime_error("dequeue: ");
66.         std::cout << "can't dequeue an empty ring" <<
  std::endl;
67.       } else {
68.         arrSize--;  // decrement size
69.         front = (front + 1) % capacity;  // set new front
70.         return arr[temp];
71.       }
72.     }
73.
74.     int16_t CircularBuffer::peek() {
75.       if (isEmpty() == true) {  // check if empty
76.         throw std::runtime_error("peek: ");
77.         std::cout << "can't peek an empty ring" <<
  std::endl;
78.       } else {
79.         return arr[front];  // peek front
80.       }
81.     }
```

PS3b: StringSound.h

```
1. #ifndef _STRING_SOUND_H
2. #define _STRING_SOUND_H
3.
4. #include <SFML/Audio.hpp>
5. #include "CircularBuffer.h"
6.
7. class StringSound {
8.  public:
9.   StringSound(double frequency); //constructor
10.       StringSound(std::vector<sf::Int16> init);
   //constructor
11.       ~StringSound(); //destructor
12.       void pluck();
13.       void tic();
14.       sf::Int16 sample();
15.       int time(); //returns counter
16.      private:
17.       CircularBuffer* circBuff; //circle buffer object
18.       int ticCounter; //for counter
19.     };
20.
21.     #endif
```

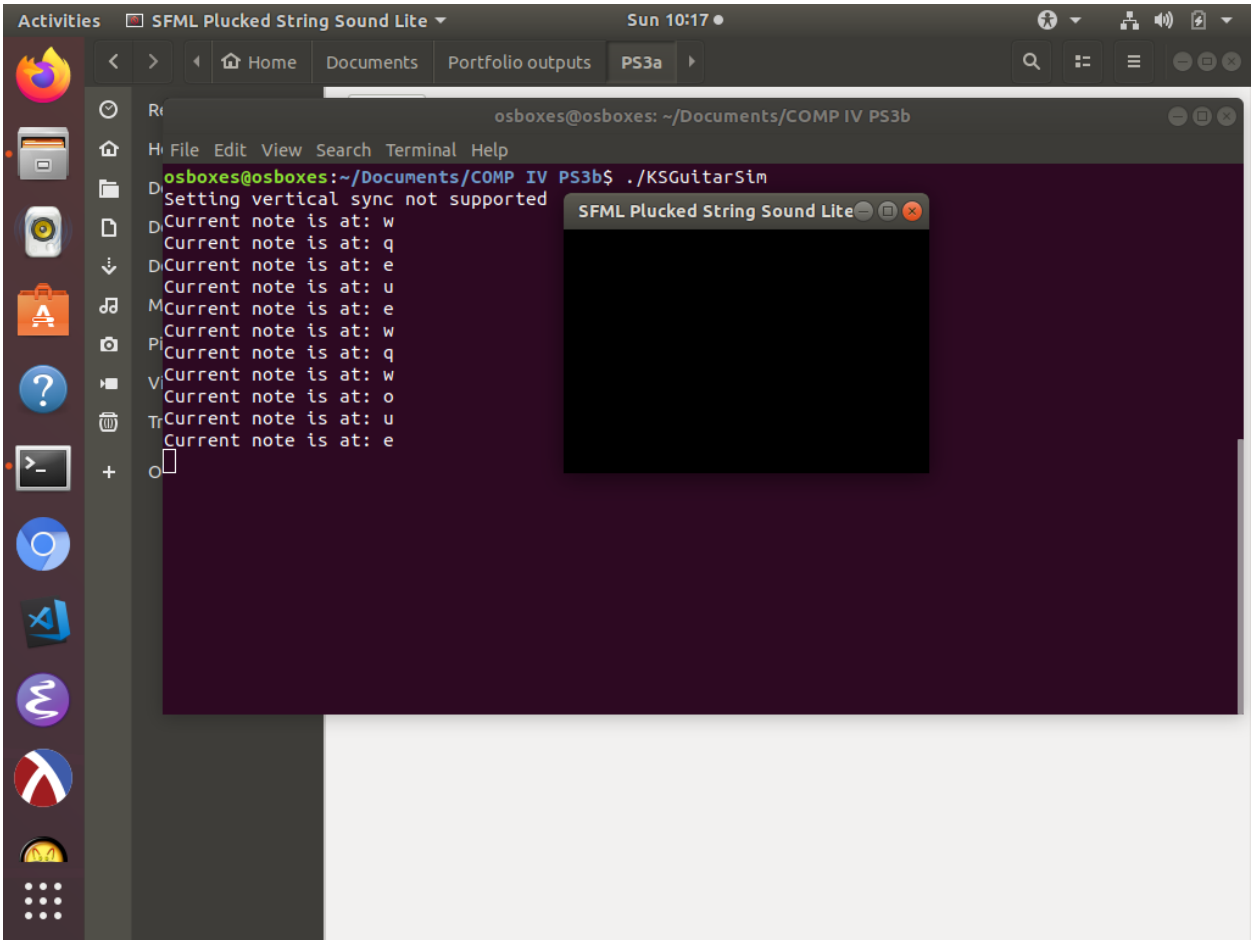
PS3b: StringSound.cpp

```
1. #include <iostream>
2. #include <exception>
3. #include <stdint.h>
4. #include <cstdlib>
5. #include <math.h>
6. #include <vector>
7. #include <SFML/Audio.hpp>
8. #include "CircularBuffer.h"
9. #include "StringSound.h"
10.
11.     using namespace std;
12.
13.     StringSound::StringSound(double frequency) {
14.       ticCounter = 0;
15.       try {
16.         circBuff = new CircularBuffer((ceil(44100 /
   frequency))); //check for valid new, create new circle
   buffer
17.       }
18.       catch (std::bad_alloc& ba) {
19.         std::cerr << "bad_alloc from constructor 1: " <<
   ba.what() << endl; //else return bad alloc
20.       }
21.     }
22.     StringSound::StringSound(std::vector<sf::Int16> init)
   {
23.       ticCounter = 0;
24.       try {
25.         circBuff = new CircularBuffer(init.size());
   //check for valid new, create new circle buffer
26.       }
27.       catch (std::bad_alloc& ba) {
28.         std::cerr << "bad_alloc from constructor 2: " <<
   ba.what() << endl; //else return bad alloc
29.       }
30.       std::vector<sf::Int16>::iterator it; //use vector
   iterator to fill circle buffer
31.       for (it = init.begin(); it != init.end(); it++) {
32.         if (circBuff->isFull() == true) { //buffer should
   not be full on creation
33.           throw std::runtime_error("Constructor 2: ");
34.           cout << "Can't enqueue full circular buffer" <<
   endl;
```

```
35.         }
36.         circBuff->enqueue(*it); //fill circle buffer
37.       }
38.     }
39.     StringSound::~StringSound() {
40.       delete circBuff; //destructor for circle buffer
41.     }
42.     void StringSound::pluck() {
43.       while (circBuff->isEmpty() == false) { //pluck
  cannot work if circle buffer is empty
44.         if (circBuff->isEmpty() == true) { //if it is
  empty throw run time error
45.           throw std::runtime_error("Pluck: ");
46.           cout << "Can't enqueue full circular buffer" <<
  endl;
47.         }
48.         circBuff->dequeue(); //dequeue plucked from circle
  buffer
49.       }
50.       while (circBuff->isFull() == false) { //if circle
  buffer is not full, enqueue rand
51.         if (circBuff->isFull() == true) {
52.           throw std::runtime_error("Pluck: ");
53.           cout << "Can't dequeue empty circular buffer" <<
  endl; //circle buffer should not be full at this point
54.         }
55.         circBuff->enqueue((sf::Int16)(rand() & 0xffff));
56.       }
57.     }
58.     void StringSound::tic() {
59.       int16_t t1 = circBuff->dequeue(); //create t1 and t2
  vars for tic
60.       int16_t t2 = circBuff->peek();
61.       circBuff->enqueue(0.5 * 0.996 * (t1 + t2));
  //enqueue tic
62.       ticCounter++; //increment counter
63.     }
64.     sf::Int16 StringSound::sample() {
65.       if (circBuff->isEmpty() == true) { //cannot peek
  empty circle buffer so throw error if empty
66.         throw std::runtime_error("Peek: ");
67.         cout << "Can't peek empty circular buffer" <<
  endl;
68.       }
69.       return circBuff->peek(); //provided for sample
70.     }
71.     int StringSound::time() {
```

```
72.       return ticCounter; //display counter
73.     }
```

PS3b: Output


Activities
SFML Plucked String Sound Lite
Sun 10:17
Home
Documents
Portfolio outputs
PS3a
osboxes@osboxes: ~/Documents/COMP IV PS3b
File Edit View Search Terminal Help
osboxes@osboxes:~/Documents/COMP IV PS3b$ ./KSGuitarSim
Setting vertical sync not supported
Current note is at: w
Current note is at: q
Current note is at: e
Current note is at: u
Current note is at: e
Current note is at: w
Current note is at: q
Current note is at: w
Current note is at: o
Current note is at: u
Current note is at: e
SFML Plucked String Sound Lite

PS3b: Discussion

What was Accomplished:

This assignment adds new functionality to the circular buffer class that was implemented in the previous assignment. In this part, the StringSound class was created to do the synthesizing of the guitar sound. The constructor created a guitar string sound with a given input frequency using a sampling rate of 44,100. Another constructor was created to accept a vector as an input parameter. To make the guitar string reverberation, a pluck function was implemented. To keep track of timing for the sound fade, a tic function was implemented to adjust the circular buffer accordingly. The program was driven by a main called KSGuitarSim.cpp. This initialized the sound buffer and allowed for everything to be controlled using keyboard input. Each note was assigned to a given keyboard input based on a piano style layout. Unfortunately my program did not produce sound. This seems to have been issue with virtual box as a lot of other students has encountered similar issues. Everything else functioned properly as proven by the many try catch blocks, exceptions, and boost tests throughout the program. All though it is not possible to prove there was sound, the output page shows the open SFML window and a running list of the keys as they are pressed to make sound.

Key Algorithms, Data Structures, OO Designs:

Since this program was an expansion of what was done in the previous assignment there was still the use of array to create a queue for the circular buffer. The enqueue and dequeue functions were designed to work using first in first out (FIFO) principles as a queue should. In addition, this program was filled with vectors. One of the constructors in the StringSound class could take in a vector as input for initializing several items into the circular buffer at a given time. Vectors were also used for the sound samples and the sound buffer to keep track of the notes and keyboard input. Similar to what was done in the CircularBuffer class, in the StringSound class encapsulation and abstraction were used to hide private variables where needed and access them using the necessary functions. Polymorphism was exercised as two constructors were implemented, one that took in a double and one that would take in a vector of int16 variables.

What was Learned:

This assignment really helped fine tune my error handling skills as it had me implement Boost tests, in function exceptions, and try catch blocks all to make sure everything was running as it should. This assignment also served to further expand my knowledge of SFML functionality. SF Int16 variables were used in the vectors to allow for the sound buffer and sound sample functionality with given keyboard input. Although I was not able to hear the outcome from all this work this assignment seemed to really solidify my knowledge of SFML, error handling, and exception testing.

PS4

DNA Sequence Alignment

Needleman-Wunsch

match = 1 mismatch = -1 gap = -1

		G	C	A	T	G	C	U
	0	-1	-2	-3	-4	-5	-6	-7
G	-1	1	0	-1	-2	-3	-4	-5
A	-2	0	0	1	0	-1	-2	-3
T	-3	-1	-1	0	2	1	0	-1
T	-4	-2	-2	-1	1	1	0	-1
A	-5	-3	-3	-1	0	0	0	-1
C	-6	-4	-2	-2	-1	-1	1	0
A	-7	-5	-3	-1	-2	-2	0	0

PS4: Makefile

```
1. CC = g++
2. CFLAGS = -std=c++17 -Wall -pedantic -Werror
3. SFMLFLAGS = -lsfml-system
4.
5. all: ED
6.
7. ED: main.o ED.o
8.         $(CC) -o ED main.o ED.o $(SFMLFLAGS)
9.
10.     main.o: main.cpp ED.h
11.             $(CC) -c main.cpp -o main.o $(CFLAGS)
12.
13.     ED.o: ED.cpp ED.h
14.             $(CC) -c ED.cpp -o ED.o $(CFLAGS)
15.
16.     clean:
17.             rm -rf ED *.o *.~
```

PS4: main.cpp

```
1. // Copyright 2020 Jacob Blumsack
2.
3. #include "ED.h"
4.
5. int main(int argc, char * argv[]) {
6.   std::string s1;  // input string 1
7.   std::string s2;  // input string 2
8.   std::string alignment;  // holds alignment
9.   sf::Clock clock;  // clock for time
10.       sf::Time t;  // holds time
11.       std::cin >> s1 >> s2;  // read in each string
12.       ED editDistance(s1, s2);  // create ED object
13.       int editDist = editDistance.OptDistance();  // calc
    edit distance
14.       // editDistance.print();
15.       alignment = editDistance.Alignment();  // calc
    alignment
16.
17.       // output edit alignment
18.       std::cout << "Edit distance is: " << editDist <<
    std::endl << alignment << std::endl;  // NOLINT
19.
20.       t = clock.getElapsedTime();  // get elapsed time
21.
22.       // output elapsed time in seconds
23.       std::cout << "Elapsed Time is: " << t.asSeconds() <<
    " Seconds" << std::endl;
24.
25.       return 0;
26.     }
```

PS4: ED.h

```
1. // Copyright 2020 Jacob Blumsack
2.
3. #ifndef _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS4_ED_H_
4. #define _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS4_ED_H_
5.
6. #include <iomanip>
7. #include <algorithm>
8. #include <iostream>
9. #include <string>
10.     #include <sstream>
11.     #include <stdexcept>
12.     #include <vector>
13.     #include <SFML/System.hpp>
14.
15.     class ED {
16.      public:
17.       ED();
18.       ED(std::string string1, std::string string2);
19.       ~ED();
20.       static int penalty(char a, char b);
21.       static int min(int a, int b, int c);
22.       int OptDistance();
23.       std::string Alignment();
24.       void print();
25.      private:
26.       std::string s1;
27.       std::string s2;
28.       std::vector<std::vector<int>> matrix;
29.     };
30.
31.     #endif  // _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS4_ED_H_
```

PS4: ED.cpp

```
1. // Copyright 2020 Jacob Blumsack
2.
3. #include <limits>
4. #include <string>
5. #include <sstream>
6. #include <iostream>
7. #include <iomanip>
8. #include <exception>
9. #include "ED.h"
10.
11.     ED::ED() {
12.       s1 = "default s1";  // default string 1
13.       s2 = "default s2";  // default string 2
14.     }
15.     ED::ED(std::string string1, std::string string2) {
16.       s1 = string1;  // set string 1
17.       s2 = string2;  // set string 2
18.     }
19.     ED::~ED() {  // destructor
20.     }
21.     int ED::penalty(char a, char b) {
22.       if (a == b) {  // 0 if equal to
23.         return 0;
24.       } else if (a != b) {  // 1 if not equal to
25.         return 1;
26.       }
27.       return -1;
28.     }
29.     int ED::min(int a, int b, int c) {
30.       int minimum = std::numeric_limits<int>::max();  //
  set min to be max initially
31.       if (a < minimum) {  // if a is min, min = a
32.         minimum = a;
33.       }
34.       if (b < minimum) {  // if b is less than min, min =
  b
35.         minimum = b;
36.       }
37.       if (c < minimum) {  // if c is less than min, min =
  c
38.         minimum = c;
39.       }
40.       return minimum;  // return min
```

```
41.     }
42.     int ED::OptDistance() {
43.       int matrixM = s1.length();  // set x val of matrix
44.       int matrixN = s2.length();  // set y val of matrix
45.       int x = 0;  // keep track of matrix x
46.       int y = 0;  // keep track of matrix y
47.       int optDistance1 = 0;  // initialize opt case 1
48.       int optDistance2 = 0;  // initialize opt case 2
49.       int optDistance3 = 0;  // initialize opt case 3
50.       for (int m = 0; m <= matrixM; m++) {  // initialize
  matrix
51.         std::vector<int> columns;  // each colomn is a
  vector
52.         matrix.push_back(columns);
53.         for (int n = 0; n <= matrixN; n++) {
54.           matrix.at(m).push_back(0);  // initialize each
  val in matrix to 0
55.         }
56.       }
57.       for (y = 0; y <= matrixN; y++) {  // initialize
  rightmost column
58.         matrix[matrixM][y] = (2 * (matrixN - y));  // 0,
  2, 4, 6...
59.       }
60.       for (x = 0; x <= matrixM; x++) {  // initialize
  bottommost row
61.         matrix[x][matrixN] = (2 * (matrixM - x));  // 0,
  2, 4, 6...
62.       }
63.       // start at second to bottommost row
64.       for (x = (matrixM - 1); x >= 0; x--) {  // decrement
  through columns
65.         // start at second to rightmost column
66.         for (y = (matrixN - 1); y >= 0; y--) {  // right
  to left through each row
67.           // calc opt distance 1
68.           optDistance1 = matrix[x + 1][y + 1] +
  penalty(s1[x], s2[y]);
69.           optDistance2 = matrix[x + 1][y] + 2;  // calc
  opt distance 2
70.           optDistance3 = matrix[x][y + 1] + 2;  // calc
  opt distance 3
71.           // min of 3 opt distances
72.           matrix[x][y] = min(optDistance1, optDistance2,
  optDistance3);
73.         }
74.       }
```

```
75.       return matrix[0][0];  // return matrix
76.     }
77.     std::string ED::Alignment() {
78.       int matrixN = s1.length();  // set x val of matrix
79.       int matrixM = s2.length();  // set y val of matrix
80.       int x = 0;  // for current x val of matrix
81.       int y = 0;  // for current y val of matrix
82.       int optDistance1 = 0;  // initialize opt case 1
83.       int optDistance2 = 0;  // initialize opt case 2
84.       int optDistance3 = 0;  // initialize opt case 3
85.       int penaltyVal = 0;  // track penalty
86.       std::ostringstream outputString;  // set up output
  string
87.       std::string stringAlign;
88.       // x = 0, y = 0, matrixN = 10, matrixM = 8
89.       while (x < matrixM - 1 || y < matrixN - 1) {
90.         penaltyVal = penalty(s1[x], s2[y]);
91.         try {  // set each opt distance
92.           optDistance1 = (matrix.at(x + 1).at(y + 1));
93.         } catch (const std::out_of_range& error) {
94.           optDistance1 = -1;
95.         }
96.         try {
97.           optDistance2 = (matrix.at(x + 1).at(y) + 2);
98.         } catch (const std::out_of_range& error) {
99.           optDistance2 = -1;
100.        }
101.        try {
102.          optDistance3 = (matrix.at(x).at(y + 1) + 2);
103.        } catch (const std::out_of_range& error) {
104.          optDistance3 = -1;
105.        }  // check each case
106.        if (s1[x] == s2[y] && matrix[x][y] ==
  optDistance1) {
107.          outputString << s1[x] << " " << s2[y] << " " <<
  penaltyVal << std::endl;
108.          x++;
109.          y++;
110.        } else if (matrix[x][y] == optDistance1 +
  penaltyVal) {
111.          outputString << s1[x] << " " << s2[y] << " " <<
  penaltyVal << std::endl;
112.          x++;
113.          y++;
114.        } else if (matrix[x][y] == optDistance2) {
115.          outputString << s1[x] << " - " << 2 <<
  std::endl;
```

```
116.           x++;
117.         } else if (matrix[x][y] == optDistance3) {
118.           outputString << " - " << s2[y] << " " << 2 <<
  std::endl;
119.           y++;
120.         } else {
121.           x++;
122.           y++;
123.         }
124.       }
125.       stringAlign = outputString.str();
126.       return stringAlign;
127.     }
128.     void ED::print() {
129.       std::vector<std::vector<int>>::iterator iter1;  //
  iterator for columns
130.       for (iter1 = matrix.begin(); iter1 != matrix.end();
  iter1++) {
131.         // lambda expression to go through rows
132.         for_each((*iter1).begin(), (*iter1).end(), [](int
  i) {
133.             // cout in grid format for readability
134.             std::cout << std::right << std::setw(3) << i
  << " ";
135.           });
136.         std::cout << std::endl;
137.         }
138.     }
```

PS4: Output

```
osboxes@osboxes:~/Documents/COMP IV PS4$ valgrind ./ED < example10.txt
==2515== Memcheck, a memory error detector
==2515== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2515== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2515== Command: ./ED
==2515==
Edit distance is: 7
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


Elapsed Time is: 0.161703 Seconds
==2515==
==2515== HEAP SUMMARY:
==2515==     in use at exit: 0 bytes in 0 blocks
==2515==   total heap usage: 71 allocs, 71 frees, 81,224 bytes allocated
```

```
==2515==
==2515== All heap blocks were freed -- no leaks are possible
==2515==
==2515== For counts of detected and suppressed errors, rerun
with: -v
==2515== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0
from 0)
```

PS4: Discussion

What was Accomplished:

In this assignment I was able to take in two different DNA sequences, calculate and display their alignment, and calculate and display their edit distance. On smaller scale, many edit distances were calculated between the two input strings. From this the penalties comparing each string are calculated through recursive means. To find the optimal alignment, the minimum of three different edit distance calculations are taken as it is incremented from character to character. This can be shown as a matrix between the two input DNA strings. The optimal alignment is then found through retracing through based on aligning character by character using the opts. In the implementation the constructor takes in the two DNA strings and initializes them in the ED class. The penalty function calculates the penalty between characters based on whether they match or not. The min function returns the minimum of three input integers (find optimal alignment). The optDistance function is where the matrix is populated and then the alignment functions is where the matrix is retraced and the optimal alignment is found as discussed above. This is then displayed in the table format as required using a print function. Once implemented, I used valgrind to compute how much space, time and memory the process uses under different circumstances. The output page shows the resulting alignment when valgrind was run on the example10.txt file. The edit distance is listed directly above the alignment.

Key Algorithms, Data Structures, OO Designs:

In this assignment, the Needleman Wunsch algorithm and the Hirschberg algorithm were both explored. The Needleman Wunsch algorithm describes the process of aligning the DNA strings through the use of the pairwise alignment on the matrix. The Hirschberg algorithm is the dynamic programming paradigm that improves upon the optimality of the Needleman Wunsch algorithm. As for data structures, a vector of vectors was used to construct the matrix. In the optDistance function, the matrix is initialiazed using the lengths of the two input DNA strings. Encapsulation and abstraction are practiced in this assignment as well for member variables in the class are private where they should be and can only be accessed by the functions that need them.

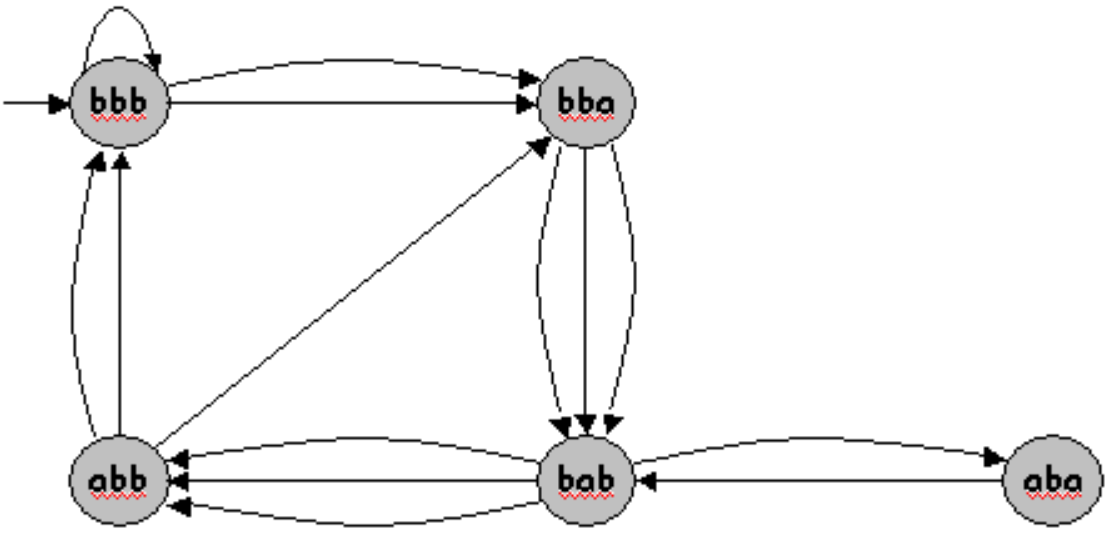
What was Learned:

Lambda functions were further expanded upon in this assignment. I used on in the print function to print part of my matrix in lieu of using a vector iterator. I never knew how to align DNA strings at all so the whole aligning process itself was a learning experience. I also had limited experience with valgrind before this assignment so it was interesting figuring out how to calculate how much memory it takes to run the program and how long it takes to do so. The massif tool proved to be very interesting as it showed an analysis of the heap after running the

program. This assignment was also a good refresher as to working with a vector of vectors. This proved to be a very efficient way to implement a matrix as was required in this assignment.

PS5

Markov Model of Natural Language



PS5: Makefile

```
1. CC = g++
2. CFLAGS = -std=c++17 -Wall -pedantic -Werror
3. Boost = -lboost_unit_test_framework
4.
5. all: TextGenerator test
6.
7. test: MModel.o test.o
8.         $(CC) -o test MModel.o test.o $(Boost)
9.
10. test.o: test.cpp MModel.h
11.         $(CC) -c test.cpp -o test.o $(CFLAGS)
12.
13. TextGenerator: MModel.o TextGenerator.o
14.         $(CC) -o TextGenerator TextGenerator.o
  MModel.o $(CFLAGS)
15.
16. TextGenerator.o: MModel.h TextGenerator.cpp
17.         $(CC) -c TextGenerator.cpp $(CFLAGS)
18.
19. MModel.o: MModel.h MModel.cpp
20. $(CC) -c MModel.cpp $(CFLAGS)
```

PS5: TextGenerator.cpp

```
1. // Copyright Jacob Blumsack 2020
2.
3. #include <map>
4. #include <string>
5. #include <sstream>
6. #include <iostream>
7. #include <cstdlib>
8. #include "MModel.h"
9.
10.     int main(int argc, char * argv[]) {
11.       srand(time(NULL));  // for rand
12.       char c;
13.       int k;  // holds arg k
14.       int l;  // holds arg l
15.       std::string inputText;  // for input text
16.       std::string kText;  // kgram text
17.       int i;
18.       if (argc != 3) {  // check number of args
19.         std::cout << "Three arguments required: K, L,
  InputText.txt" << std::endl;
20.         return 1;
21.       }
22.       k = atoi(argv[1]);  // convert input to ints
23.       l = atoi(argv[2]);
24.       if (l < k) {  // make sure l is >= k
25.         std::cout << "L must be greater than or equal to
  K" << std::endl;
26.         return 1;
27.       }
28.       std::cin.get(c);
29.       while (!std::cin.eof()) {
30.         inputText.push_back(c);
31.         std::cin.get(c);
32.       }
33.       MModel mmodel(inputText, k);
34.       for (i = 0; i < k; i++) {
35.         kText.push_back(inputText[i]);
36.       }
37.       std::cout << mmodel.generate(kText, l) << std::endl;
38.       std::cout << mmodel << std::endl;
39.       return 0;
40.     }
```

PS5: MModel.h

```
1. // Copyright Jacob Blumsack 2020
2.
3. #ifndef _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS5_MMODEL_H_
4. #define _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS5_MMODEL_H_
5.
6. #include <cstdlib>
7. #include <map>
8. #include <iostream>
9. #include <algorithm>
10.     #include <string>
11.     #include <stdexcept>
12.     #include <vector>
13.
14.     class MModel {
15.      public:
16.       MModel(std::string text, int k);
17.       int kOrder();
18.       int freq(std::string kgram);
19.       int freq(std::string kgram, char c);
20.       char kRand(std::string kgram);
21.       std::string generate(std::string kgram, int L);
22.       friend std::ostream& operator << (std::ostream& out,
    MModel& mmodel);
23.      private:
24.       int kOrdered;
25.       std::string kAlphabet;
26.       std::string kInitial;
27.       std::map <std::string, std::map <char, int>>
    kGramFreq;
28.       std::map <std::string, int> kGram;
29.     };
30.
31.     template<typename m, typename n>
32.     void printKGram(std::map <m, n> const &map) {
33.       std::for_each(map.begin(), map.end(), [](const
    std::pair<std::string, int> &p) {  // NOLINT
34.           std::cout << "KGRAM+1 " << p.first << " : " <<
    p.second << std::endl;
35.         });
36.     }
37.
38.     #endif  //
    _HOME_OSBOXES_DOCUMENTS_COMP_IV_PS5_MMODEL_H_
```

PS5: MModel.cpp

```
1. // Copyright Jacob Blumsack 2020
2.
3. #include <map>
4. #include <stdexcept>
5. #include <string>
6. #include <cstdlib>
7. #include <iostream>
8. #include <vector>
9. #include <ctime>
10.     #include "MModel.h"
11.
12.     MModel::MModel(std::string text, int k) {
13.       unsigned length  = text.size();
14.       kOrdered = k;  // set kOrder to k
15.       kInitial = text;  // initial k is text
16.       std::string kgram;  // string for kgram map
    populating
17.       char kChar;
18.       unsigned m;  // loop iterator
19.       unsigned i;  // iterator for for loop
20.       int kCount = 0;  // counts
21.       if ((unsigned)k >= length) {
22.         throw std::runtime_error("text should be at least
    of length k");
23.       }
24.       for (m = 0; m < length; m++) {
25.         if (kAlphabet.find(text[m]) == std::string::npos)
    {
26.           // initialize kAlphabet
27.           kAlphabet.push_back(text[m]);
28.         }
29.         // make sure string is clear
30.         kgram.clear();
31.         i = m;
32.         kCount = 0;  // reset kCount
33.         while (kCount < kOrdered) {
34.           // push char to kgram
35.           kgram.push_back(text[i]);
36.           // increment kCount
37.           kCount++;
38.           if (i == length - 1) {
39.             i = 0;
40.           } else {
```

```
41.             i++;
42.           }
43.         }
44.         kChar = text[i];
45.         // check if kgram is populated
46.         if (kGram.find(kgram) == kGram.end()) {
47.           kGram[kgram] = 1;
48.           kGramFreq[kgram][kChar] = 1;
49.         } else {  // else continue
50.           kGram[kgram] += 1;
51.           // check kGramFrq is populated
52.           if (kGramFreq[kgram].find(kChar) ==
  kGramFreq[kgram].end()) {
53.             kGramFreq[kgram][kChar] = 1;
54.           } else {
55.             kGramFreq[kgram][kChar] += 1;
56.           }
57.         }
58.       }
59.     }
60.     int MModel::kOrder() {
61.       return kOrdered;  // return kOrder
62.     }
63.     int MModel::freq(std::string kgram) {
64.       // throw error if kgram length is wrong
65.       if (kgram.size() != (unsigned)kOrdered) {
66.         throw std::runtime_error("kgram is not of length
  k");
67.       }
68.       if (kOrdered == 0) {
69.         return kInitial.size();  // return initial size if
  no kgram
70.       } else if (kGram.find(kgram) != kGram.end()) {
71.         return kGram[kgram];  // return number kgram in
  kGram map
72.       } else {
73.         return 0;  // no frequency: return 0
74.       }
75.     }
76.     int MModel::freq(std::string kgram, char c) {
77.       // throw error if kgram length is wrong
78.       if (kgram.size() != (unsigned)kOrdered) {
79.         throw std::runtime_error("kgram is not of length
  k");
80.       }
81.       // throw error if no such kgram
82.       if (kGram.find(kgram) == kGram.end()) {
```

```
83.         throw std::runtime_error("No such kgram");
84.       }
85.       // use find() to check for occurences of c
86.       if (kGramFreq[kgram].find(c) ==
  kGramFreq[kgram].end()) {
87.         return 0;  // if none return 0
88.       }
89.       // else return number of occurences
90.       return kGramFreq[kgram][c];
91.     }
92.     char MModel::kRand(std::string kgram) {
93.       std::string followingChar;  // string of all chars
  that follow kgram
94.       char randomChar;  // random char from followingChar
  string
95.       int j;  // iterator for for loop
96.       // iterator for kGramFreq
97.       auto it1 = kGramFreq.find(kgram);
98.       // iterator for map in kGramFreq
99.       auto it2 = it1->second;
100.      // throw error if kgram length is wrong
101.      if (kgram.size() != (unsigned)kOrdered) {
102.        throw std::runtime_error("kgram is not of length
  k");
103.      }
104.      // find() returns value of end() if no kgram is
  found
105.      if (kGram.find(kgram) == kGram.end()) {  // throw
  error if no kgram
106.        throw std::runtime_error("No such kgram");
107.      }
108.      // use for loop to check next character
109.      for (auto it3 = it2.begin(); it3 != it2.end();
  it3++) {
110.        for (j = 0; j < it3->second; j++) {
111.          followingChar.push_back(it3->first);
112.        }
113.      }
114.      // use rand() to select rand char following kgram
115.      randomChar = followingChar[rand() %
  followingChar.size()];
116.      return randomChar;  // return rand char
117.    }
118.    std::string MModel::generate(std::string kgram, int L)
  {
119.      std::string genString = kgram;  // run kRand on
  genString
```

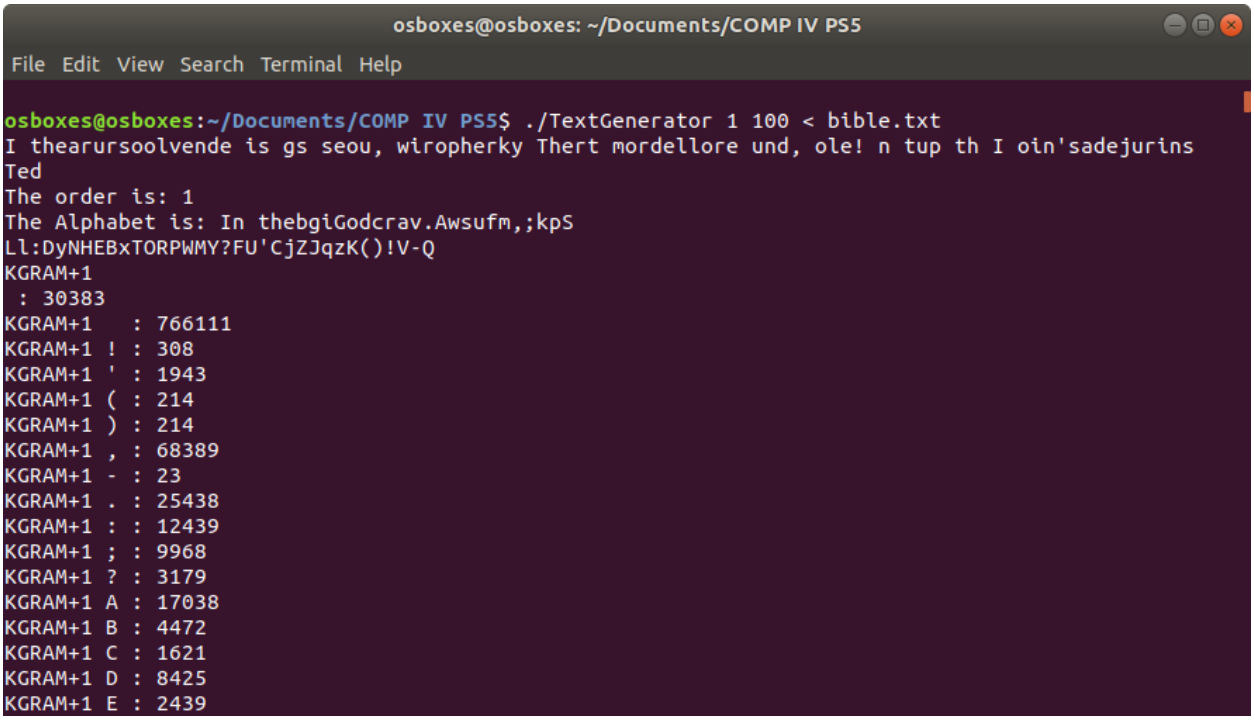
```
120.      std::string kString = kgram;  // first k chars is
  kgram
121.      char randomChar;  // holds random char from kRand
122.      unsigned x;  // iterates for for loop
123.      // throw error if kgram length is wrong
124.      if (kgram.size() != (unsigned)kOrdered) {
125.        throw std::runtime_error("kgram is not of length
  k");
126.      }
127.      for (x = kgram.size(); x <
  static_cast<unsigned>(L);) {
128.        if (freq(kString) != 0) {
129.          randomChar = kRand(kString);  // generate random
  char from kgram
130.          // push kgram to be first k chars of genString
131.          genString.push_back(randomChar);
132.          kString.erase(kString.begin());  // reset
  kString
133.          kString.push_back(randomChar);  // set kString
  to new kgram
134.          x++;
135.        } else {
136.          randomChar = kAlphabet[rand() %
  kAlphabet.size()];
137.          kString.erase(kString.begin());  // reset
  kString
138.          kString.push_back(randomChar);  // set kString
  to new kgram
139.        }
140.      }
141.      return genString;  // return string
142.    }
143.    std::ostream& operator << (std::ostream& out, MModel
  &mmodel) {
144.      MModel temp(mmodel.kInitial, (mmodel.kOrdered + 1));
145.      // iterator for map
146.      std::map <std::string, int>::iterator kIt;
147.      std::map <std::string, int>::iterator kIt2;
148.      out << "The order is: " << mmodel.kOrdered <<
  std::endl;
149.      out << "The Alphabet is: " << mmodel.kAlphabet <<
  std::endl;
150.      // lambda function for printing kgram
151.      printKGram(mmodel.kGram);
152.      // use iterator and for loop to output k+1gram
153.      for (kIt2 = temp.kGram.begin(); kIt2 !=
  temp.kGram.end(); kIt2++) {
```

```
154.        out << "KGRAM+1 " << kIt2->first << " : " <<
  kIt2->second << std::endl;
155.      }
156.      return out;
157.    }
```

PS5: test.cpp

```
1. // CopyRight Jacob Blumsack 2020
2. #define BOOST_TEST_DYN_LINK
3. #define BOOST_TEST_MODULE Main
4.
5. #include <iostream>
6. #include <map>
7. #include <string>
8. #include <stdexcept>
9. #include <exception>
10.     #include "MModel.h"
11.     #include <boost/test/unit_test.hpp>
12.
13.     BOOST_AUTO_TEST_CASE(MModel_CONSTRUCTOR) {
14.       // test text must be at least length k
15.       BOOST_REQUIRE_THROW(MModel("ab", 3),
   std::runtime_error);
16.     }
17.     BOOST_AUTO_TEST_CASE(MModel_KORDER) {
18.       // test kOrdered is set to k
19.       MModel mmodel("abaabb", 3);
20.       BOOST_REQUIRE(mmodel.kOrder() == 3);
21.     }
22.     BOOST_AUTO_TEST_CASE(MModel_FREQ1) {
23.       // test frequency
24.       MModel mmodel("abaabb", 1);
25.       BOOST_REQUIRE(mmodel.freq("a") == 1);
26.     }
27.     BOOST_AUTO_TEST_CASE(MModel_FREQ2) {
28.       // test frequency
29.       MModel mmodel("abaabb", 2);
30.       BOOST_REQUIRE(mmodel.freq("ab", 'a') == 1);
31.     }
32.     BOOST_AUTO_TEST_CASE(MModel_KRAND) {
33.       // test krand returns character from alphabet
34.       MModel mmodel("abaabb", 2);
35.       char kRandChar = mmodel.kRand("ab");
36.       BOOST_REQUIRE(kRandChar == 'a' || kRandChar == 'b');
37.     }
38.     BOOST_AUTO_TEST_CASE(MModel_GENERATE) {
39.       // test generate is of correct length
40.       MModel mmodel("abaabb", 2);
41.       std::string genString = mmodel.generate("ab", 10);
42.       BOOST_REQUIRE(genString.size() == 10); }
```

PS5: Output


osboxes@osboxes: ~/Documents/COMP IV PS5
File Edit View Search Terminal Help
osboxes@osboxes:~/Documents/COMP IV PS5$ ./TextGenerator 1 100 < bible.txt
I thearursoolvende is gs seou, wiropherky Thert mordellore und, ole! n tup th I oin'sadejurins
Ted
The order is: 1
The Alphabet is: In thebgiGodcrav.Awsufm,;kpS
Ll:DyNHEBxTORPWMY?FU'CjZJqzK()!V-Q
KGRAM+1
: 30383
KGRAM+1 : 766111
KGRAM+1 ! : 308
KGRAM+1 ' : 1943
KGRAM+1 (: 214
KGRAM+1) : 214
KGRAM+1 , : 68389
KGRAM+1 - : 23
KGRAM+1 . : 25438
KGRAM+1 : : 12439
KGRAM+1 ; : 9968
KGRAM+1 ? : 3179
KGRAM+1 A : 17038
KGRAM+1 B : 4472
KGRAM+1 C : 1621
KGRAM+1 D : 8425
KGRAM+1 E : 2439

PS5: Discussion

What was Accomplished:

A Markov model of natural language predicts characters with probabilities based on the current state of the string. In this assignment's implementation the MModel constructor creates a markov model of a given input k from an input text string. This was done so using two maps. One to keep track of the frequency of a given character in the string and one to mark the frequency of the kgram. A kRand function was implemented to returna random character that must have been a character that followed the kgram in the original string. Then using the generate function, a string of a given input length would be randomly generated. The first k characters of this new string must be the kgram, the krand function is continuously called to generate the rest of the string until the input length that was specified. The program was driven by TextGenerator.cpp which would take in the arguments and create the mmodel objects. The executable had to take in the arguments for k length of kgram, l length of the new string and a file where the original string would be derived from. The class functions were all tested using the boost test network. This was all accomplished as specified. The output page shows a capture of the result. In this image a pseudo random string from the text generator can be seen. Then the order and the alphabet are displayed. Lastly a portion of the list of k+1grams is shown. Both the kgrams and k+1grams were captured however this is a very long list and cannot be shown in one image.

Key Algorithms, Data Structures, OO Designs:

The main structure that this program was built off of was the maps. These were crucial to storing the frequencies of the characters in the string and the kgram frequencies. For the character frequencies, a map with string key and an integer value. The kgram frequency map had a string key and the value was another map with a char key and an integer value. A template was used in this assignment as well. The template functioned as a pass through parameter in a print function that was made to print the k+1grams. The print function used a lambda expression to print the first and second elements values throughout the map. The ED class implemented in this assignment also demonstrated encapsulation and abstraction as the private variables were kept abstracted and only accessed by the member functions that needed to reference them. Polymorphism was demonstrated through the overloading of the output operator. This added the extra functionality of being able to output the mmodel object to display its order, alphabet, kgrams, and k+1grams.

What was Learned:

This assignment was a really good refresher on maps and the different functions that accompany them. I had never used a map whose key references another map as a value so that

was really interesting to implement for the kgram frequencies. I learned about functions such as the find function that proved to be very helpful in use with the maps. The find function allowed for the ability to check the number occurrences of kgrams among strings such as what was done in the frequency function. I also deepened my understanding of lambda expression in this assignment. It was difficult finding a spot to implement the lambda expression for the extra credit. Using a template proved to be an unnecessarily long work around to simply using a map iterator as I did for the kgrams.

PS6

Kronos Time Clock

PS6: Makefile

```
1. CC = g++
2. CFLAGS = -std=c++17 -Wall -pedantic -Werror
3. Boost = -lboost_regex -lboost_date_time
4.
5. all: ps6
6.
7. ps6: PS6.cpp
8.         $(CC) -o ps6 PS6.cpp $(CFLAGS) $(Boost)
9.
10. clean:
11. rm -f ps6 *~
```

PS6: PS6.cpp

```
1. // Copyright 2020 Jacob Blumsack
2.
3. #include <cstdlib>
4. #include <iostream>
5. #include <vector>
6. #include <fstream>
7. #include <string>
8. #include <boost/regex.hpp>
9. #include "boost/date_time/gregorian/gregorian.hpp"
10.     #include "boost/date_time/posix_time/posix_time.hpp"
11.
12.     using boost::gregorian::date;
13.     using boost::gregorian::from_simple_string;
14.     using boost::gregorian::date_period;
15.     using boost::gregorian::date_duration;
16.
17.     using boost::posix_time::ptime;
18.     using boost::posix_time::time_duration;
19.
20.     using boost::regex;
21.
22.     template <typename T>
23.     int to_int(const T& t) {
24.       return atoi(t.str().c_str());
25.     }
26.
27.     int main(int argc, char * argv[]) {
28.       // regex for date
29.       std::string dateString("([0-9]{4})-([0-9]{1,2})-([0-
  9]{1,2}) ");
30.       // regex for time
31.       std::string timeString("([0-9]{2}):([0-9]{2}):([0-
  9]{2})");
32.       // regex for boot
33.       std::string bootString("(.*log.c.166.*)");
34.       // regex for success
35.       std::string
  successString("(.*oejs.AbstractConnector:Started
  SelectChannelConnector.*)");  // NOLINT
36.       boost::smatch sm;  // holds match for boot and
  success
37.       boost::smatch sm2;  // match for date
38.       boost::smatch sm3;  // match for time
```

```
39.       std::string str;  // store strings from files
40.       int currentLine = 1;  // keep track of current line
41.       bool bootState = true;  // track boot state
42.       // time variables
43.       ptime time1;  // for time duration
44.       ptime time2;
45.       std::string outputName(std::string(argv[1]) +
  ".rpt");
46.       // declaring input and output files
47.       std::ofstream output;
48.       std::ifstream input(argv[1], std::ifstream::in);
49.       // check that it is a valid file
50.       if (!input.is_open()) {
51.         std::cout << "File " << argv[1] << " cannot be
  opened" << std::endl;
52.         return 0;
53.       }
54.       output.open(outputName.c_str());
55.       // create regex objects for boot and success
56.       regex regexDate(dateString);  // regex for date
57.       regex regexTime(timeString);  // regex for time
58.       regex regexBoot(bootString);  // regex for boot
  message
59.       regex regexSuccess(successString);  // regex for
  success
60.
61.       while (getline(input, str)) {
62.         if (regex_search(str, sm, regexBoot)) {
63.           regex_search(str, sm2, regexDate);
64.           regex_search(str, sm3, regexTime);
65.           // create time and date objects
66.           date Date(from_simple_string(sm2[0]));
67.           ptime Time(Date, time_duration(to_int(sm3[1]),
  to_int(sm3[2]), to_int(sm3[3])));  // NOLINT
68.           time1 = Time;  // track time
69.           if (bootState) {
70.             output << "**** Boot in process ****" <<
  std::endl;
71.           }
72.           // output information to file
73.           output << "**** Device Booted Up ****" <<
  std::endl;
74.           output << sm[1] << " ";
75.           output << "[Device Boot Start]" << std::endl;
76.           if (bootState == false) {
77.             output << "**** Boot Failure ****" <<
  std::endl;
```

```
78.             bootState = true;
79.           } else {
80.             bootState = false;
81.           }
82.         } else if (regex_search(str, sm, regexSuccess)) {
83.           regex_search(str, sm2, regexDate);
84.           regex_search(str, sm3, regexTime);
85.           // create time and date objects
86.           date Date(from_simple_string(sm2[0]));
87.           ptime Time(Date, time_duration(to_int(sm3[1]),
  to_int(sm3[2]), to_int(sm3[3])));  // NOLINT
88.           time2 = Time;  // track time
89.           // calculate time duration
90.           time_duration timeDuration = (time2 - time1);
91.           // output information to file
92.           output << currentLine << "(" << argv[1] << "):
  \n";
93.           output << sm[1] << " ";
94.           output << "[Device Boot Success]" << std::endl;
95.           // output boot time
96.           output << "Total Boot Time: ";
97.           output << timeDuration.total_milliseconds() << "
  milliseconds\n" << std::endl;  // NOLINT
98.           bootState = true;
99.         }
100.        currentLine++;
101.      }
102.      return 0;
103.    }
```

PS6: Output

```
**** Boot in process ****

**** Device Booted Up ****

2014-03-25 19:11:59: (log.c.166) server started  [Device Boot Start]

435759(device1_intouch.log):

2014-03-25 19:15:02.369:INFO:oejs.AbstractConnector:Started
SelectChannelConnector@0.0.0.0:9080 [Device Boot Success]

Total Boot Time: 183000 milliseconds


**** Boot in process ****

**** Device Booted Up ****

2014-03-25 19:29:59: (log.c.166) server started  [Device Boot Start]

436859(device1_intouch.log):

2014-03-25 19:32:44.036:INFO:oejs.AbstractConnector:Started
SelectChannelConnector@0.0.0.0:9080 [Device Boot Success]

Total Boot Time: 165000 milliseconds


**** Boot in process ****

**** Device Booted Up ****

2014-03-25 22:01:46: (log.c.166) server started  [Device Boot Start]

440791(device1_intouch.log):

2014-03-25 22:04:27.514:INFO:oejs.AbstractConnector:Started
SelectChannelConnector@0.0.0.0:9080 [Device Boot Success]

Total Boot Time: 161000 milliseconds


**** Boot in process ****

**** Device Booted Up ****

2014-03-26 12:47:42: (log.c.166) server started  [Device Boot Start]

441216(device1_intouch.log):
```

```
2014-03-26 12:50:29.824:INFO:oejs.AbstractConnector:Started
SelectChannelConnector@0.0.0.0:9080 [Device Boot Success]

Total Boot Time: 167000 milliseconds


**** Boot in process ****

**** Device Booted Up ****

2014-03-26 20:41:34: (log.c.166) server started  [Device Boot Start]

442432(device1_intouch.log):

2014-03-26 20:44:13.235:INFO:oejs.AbstractConnector:Started
SelectChannelConnector@0.0.0.0:9080 [Device Boot Success]

Total Boot Time: 159000 milliseconds


**** Boot in process ****

**** Device Booted Up ****

2014-03-27 14:09:01: (log.c.166) server started  [Device Boot Start]

443411(device1_intouch.log):

2014-03-27 14:11:42.500:INFO:oejs.AbstractConnector:Started
SelectChannelConnector@0.0.0.0:9080 [Device Boot Success]

Total Boot Time: 161000 milliseconds
```

PS6: Discussion

What was Accomplished:

This assignment revolved around the use of regular expressions to analyze the int touch logs of the kronos time clock and create a text file to show every time the device was restarted in chronological order. Each time the device boots up, there is a certain string that goes through the log so a regex was made to find every time the device restarted. Another string is shown every time the boot sequence is completed so another regex was made to determine when the end of each boot sequence. Two more regexes were implemented as well, one to find the date of each boot sequence and one to determine the time of each boot sequence. Using the boost regex library, the smatch variables were created to hold the matches for each regex as each file was scanned through. At the same time I had to keep track of the pairs of startup and success messages for if a startup message came after another startup message that would mean there was a boot failure. The regex_search function would take in the input string, a match variable, and the regex to find the occurrence matching the regex in the input string. Using the regexes, the in touch logs were deciphered and the necessary information was reported out into a new text file. Boost posix time was used to determine the amount of time elapsed for each boot sequence as well. The output page shows the boot sequences captured from one of the in touch logs that was run.

Key Algorithms, Data Structures, OO Designs:

The regular expressions were crucial to the functionality of this assignment. They could search for anything from an exact string of characters to a range of dates following a specific format as was done in the assignment. The regex string ([0-9]{4})-([0-9]{1,2})-([0-9]{1,2}) would search for a string in the format of xxxx-xx-xx where each would be a digit between 0 and 9. This regex was used to find the dates. Regular expressions proved to be very versatile in their use in the assignment. No classes were implemented in this assignment as it was just a main driver.

What was Learned:

I really enjoyed learning about regular expression in this assignment. I felt they were really versatile and seemed as though they could be used for a wide range of tasks. They were very easy to catch on to as well as their conventions are fairly straight forward. In this assignment I also learned to use boost posix time for the sake of finding the elapsed time between two events. To implement the regular expressions, boost regex was used. This came with the functionality of being able to search for a regex and place the match into a boost smatch variable. This made displaying corresponding data from each log a lot easier.