

Matthew Lorette Anaya
COMP IV Sec 203: Project Portfolio
Spring 2022

Contents:

PS0 Hello World with SFML
PS1 Linear Feedback Shift Register and Image Encoding
PS2 N-Body Simulation
PS3 Recursive Graphics
PS4 Synthesizing a Plucked String Sound
PS5 DNA Sequence Alignment
PS6 Random Writer
PS7 Kronos – Intro to Regular Expression
Time to complete: 10 hours

PS0 Hello World with SFML

Setting up a build environment on a virtual Linux system and utilizing it to demonstrate the Simple and Fast Multimedia Libraries was the first job (SFML). We were to alter the SFML demo code to have it create our own sprite and respond to keyboard input when we got Linux up and running.

Key Concepts

Setting up a build environment on a virtual Linux system and utilizing it to demonstrate the Simple and Fast Multimedia Libraries was the first job (SFML). We were to alter the SFML demo code to have it create our own sprite and respond to keyboard input when we got Linux up and running. There was just one main function in this code, and it was not designed in an object-oriented manner. Because the primary goal of this project was to familiarize myself with the Linux development environment and SFML, no notable algorithms or data structures were employed.

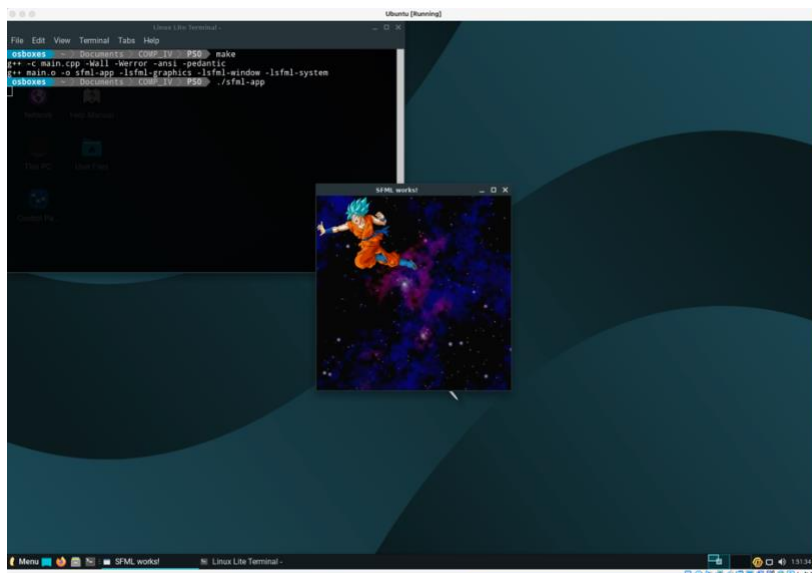
What I accomplished

I started with the SFML instructional website's sample code, which produces a window with a green circle drawn in the center when compiled and run. This project required me to construct a depiction of one of my favorite anime characters in space. I started by making the window 400x400, after which loading in my own sprite images to replace the green circle with a Goku sprite and a space like image. When you hit a directional key, the Goku sprite will travel in that direction.

What I Learned

I learned a great deal about Linux and how to compile programs using Makefiles and the Bash shell, along with setting up a virtual machine. I also learned about SFML and how it can be used to create cross-platform programs with rudimentary graphics, event handling, and audio.

Output



Makefile**Fri Apr 29 13:41:58 2022****1**

```
1: CC = g++
2:
3: all:
4:      $(CC) -c main.cpp -Wall -Werror -ansi -pedantic
5:      $(CC) main.o -o sfml-app -lsfml-graphics -lsfml-window -lsfml-sys
tem
6:
7: clean:
8:      rm *.o sfml-app *~
```

```
1: #include <SFML/Graphics.hpp>
2: #include <iostream>
3:
4: int main() {
5:
6:     // Make window
7:     sf::RenderWindow window(sf::VideoMode(400, 400), "SFML works!");
8:
9:     // Load sprite
10:    sf::Texture texture;
11:    if(!texture.loadFromFile("sprite.png"))
12:        return EXIT_FAILURE;
13:    sf::Sprite sprite(texture);
14:
15:    // Background
16:    sf::Texture star_texture;
17:    if (!star_texture.loadFromFile("starfield.jpg"))
18:        return -1;
19:    sf::Sprite background(star_texture);
20:
21:    while(window.isOpen()) {
22:        sf::Event event;
23:        while(window.pollEvent(event)) {
24:            if(event.type == sf::Event::Closed)
25:                window.close();
26:        }
27:
28:        window.clear();
29:
30:        float offsetX = 0;
31:        float offsetY = 0;
32:
33:        // Get Sprite's current position
34:        sf::Vector2f pos = sprite.getPosition();
35:
36:        // Move image around screen as long as to not move it off)
37:        if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left) && pos.x != 0)
38:            offsetX = -1;
39:        else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right) && pos.x != 4
00 - 198)
40:            offsetX = 1;
41:        else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up) && pos.y != 0)
42:            offsetY = -1;
43:        else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down) && pos.y != 40
0 - 152)
44:            offsetY = 1;
45:        else if(sf::Keyboard::isKeyPressed(sf::Keyboard::R)) {
46:            sprite.setPosition(0, 0);
47:            pos.x = pos.y = 0;
48:        }
49:        else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
50:            window.close();
51:
52:        // Set a new position
53:        sprite.setPosition(pos.x + offsetX, pos.y + offsetY);
54:
55:        // Draw images
56:        window.draw(background);
57:        window.draw(sprite);
58:
59:        window.display();
60:    }
61:
62:    return 0;
63: }
```

PS1 Linear Feedback Shift Register and PhotoMagic

PS1a and PS1b were the two halves of this project. PS1a required us to create a Fibonacci linear feedback shift register (LFSR) that would generate pseudo-random bits for use in PS1b picture encryption. After shifting each bit once to the left, the LFSR had to take a 16-bit binary seed and XOR the designated tap bits from its current state to create the new rightmost bit of its next state. PS1b would then have a main function that would accept an image file as input, encrypt its pixel data using the LFSR, and output the result.

Key Concepts

The FibLFSR class is a container for a dynamic array with a size equal to the length of the binary seed provided to it during creation; however, if the string is not precisely 16 bits long, an error is thrown. Any exceptions thrown by FibLFSR will be handled by the PhotoMagic class. The required copy/move/destructor functions are provided in the class since FibLFSR has dynamically allocated memory.

What I accomplished

PS1a: FibLFSR stores a series of shifted bits to the left and acquires the rightmost bit using tap bits. To store the provided seed, I utilized a dynamic array of integers (input bits). The class provides a generate() function that steps through the LFSR using step() as a helper function. I utilized the Boost C++ libraries to execute test cases on the FibLFSR class for this section of the assignment, which is specified in test.cpp. I double-checked that both the step() and create() methods returned the appropriate results, and I looked for an error if the given seed was shorter or longer than the register's needed 16 bits.

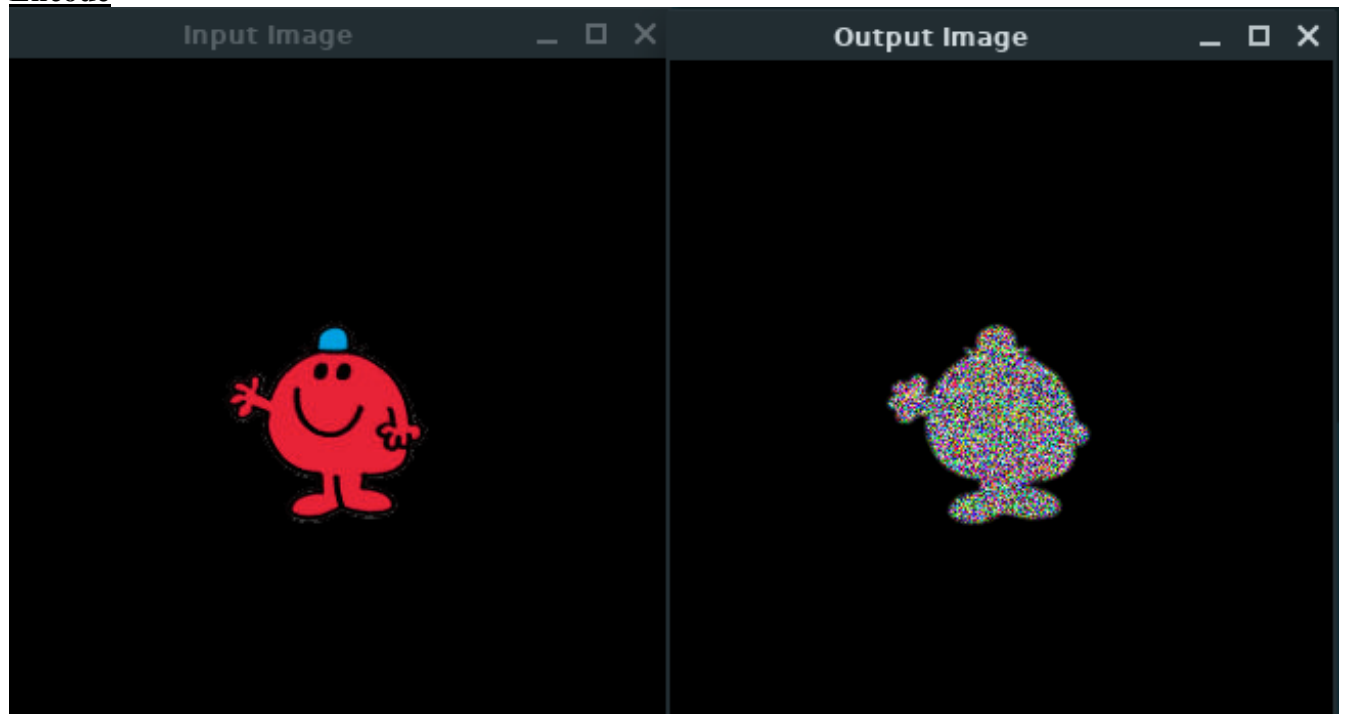
PS1b: The input filename, output filename, and a 16-bit binary seed are all required command line parameters for PhotoMagic. After entering the command, a FibLFSR object is formed, and the transform() function on the picture is invoked, which encrypts or decrypts images by XORing pixel data with register bits. The software additionally double-checks that all command-line options are accurate. Two new windows will appear in the SFML display loop, displaying both the original and the encrypted/decrypted picture.

What I Learned

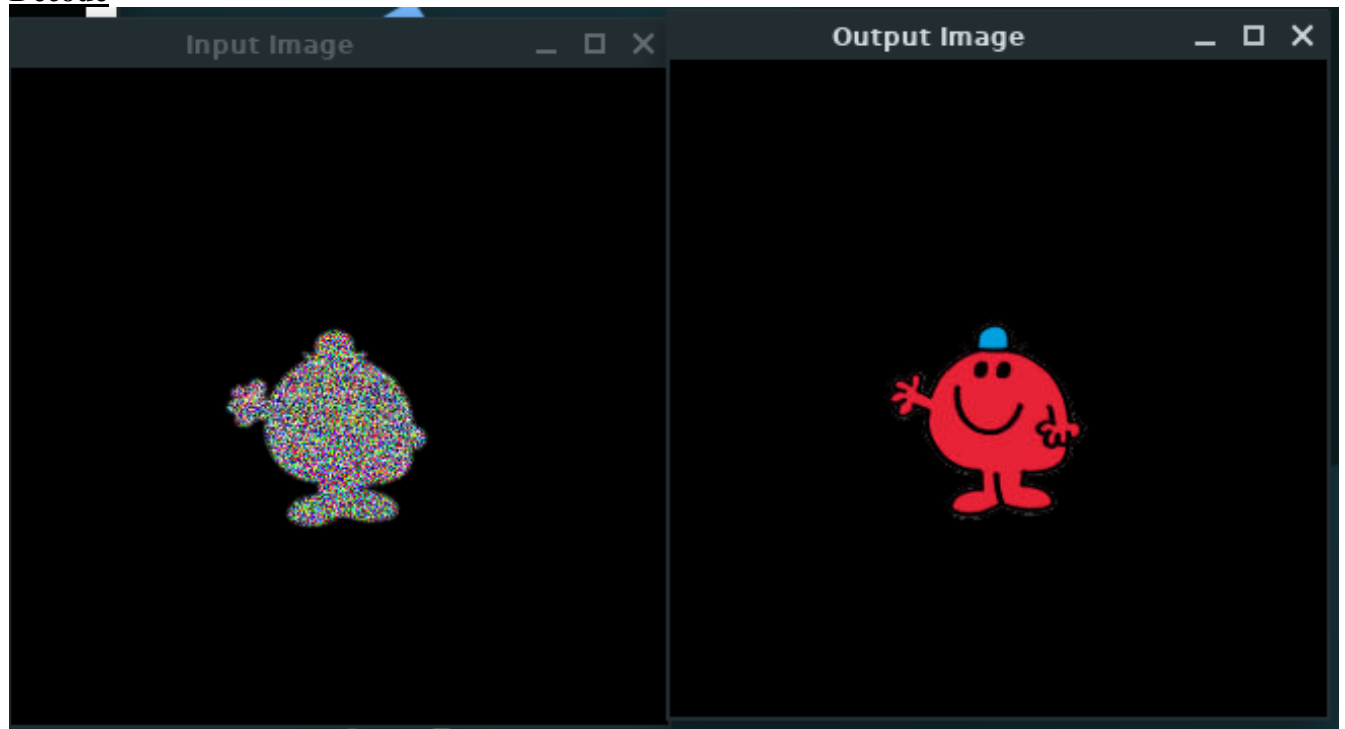
This project provided a fantastic opportunity for me to revisit some of the C++ programming methods I had previously studied, such as the rule of 5, operator overloading, exception handling, and SFML. In addition, while my FibLFSR class was still in development, I learnt how to leverage the Boost libraries to quickly perform unit tests on it. Unit test are my biggest take away from this assignment, as it also helped me at work developing them to test validation annotations on some of the DTOs I've been reconstructing.

Output

Encode



Decode



```
1: CC= g++
2: CFLAGS= -Wall -Werror -ansi -pedantic
3: SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
4:
5: all:
6:     make PhotoMagic
7:
8: PhotoMagic: PhotoMagic.o FibLFSR.o
9:     $(CC) PhotoMagic.o FibLFSR.o -o PhotoMagic $(SFMLFLAGS)
10:
11: PhotoMagic.o: PhotoMagic.cpp FibLFSR.h
12:     $(CC) $(CFLAGS) -c PhotoMagic.cpp FibLFSR.h
13:
14: FibLFSR.o: FibLFSR.cpp FibLFSR.h
15:     $(CC) -c FibLFSR.cpp
16:
17: clean:
18:     $(RM) *.o
19:     $(RM) PhotoMagic
```

```
1: /*
2: Computing IV - Assignment - PS1a + b
3: Instructor: Prof. Yelena Rykalova
4: Due Date: 02/07/22
5: Author: Matthew Lorette Anaya
6: Description: This program is an implementation of a Fibonacci Linear Feed
back      Shift Register
7:      This is the implementation of the PhotoMagic.class which tak
es thre   e arguments an input image an output image and a seed. The p
rogram    uses the the seed to encode the input image and display it a
s the o   utput image.
8: */
9: #include <iostream>
10: #include <string>
11: #include <sstream>
12: #include <SFML/System.hpp>
13: #include <SFML/Window.hpp>
14: #include <SFML/Graphics.hpp>
15: #include "FibLFSR.h"
16:
17: void transform( sf::Image& img, FibLFSR* bit_generator) {
18:     // randomize the bits in the image
19:     sf::Vector2u imgsize = img.getSize();
20:     // initialize an SFML pixel
21:     sf::Color p;
22:
23:     for(int x = 0; x < (signed)imgsize.x; x++) {
24:         for(int y = 0; y < (signed)imgsize.y; y++) {
25:             // get the current pixel from the input image
26:             p = img.getPixel(x, y);
27:
28:             // generate encoded pixels
29:             p.r = p.r ^ bit_generator -> generate(8);
30:             p.g = p.g ^ bit_generator -> generate(8);
31:             p.b = p.b ^ bit_generator -> generate(8);
32:
33:             // edit the image in-place with new encoded pixels
34:             img.setPixel(x, y, p);
35:         }
36:     }
37: }
38: int main(int argc, char* argv[]) {
39:     if(argc != 4) {
40:         std::cout << "Incorrect Input Format" << std::endl
41:             << "Input should be as follows: ./PhotoMagic <inputfilename
> <outputfilename> <seed>\n";
42:         return -1;
43:     }
44:
45:     // store input in variables
46:     std::string input_fname(argv[1]);
47:     std::string output_fname(argv[2]);
48:     std::string seed = argv[3];
49:
50:     // create an LSFR object
51:     FibLFSR bit_generator(seed);
52:
53:     // load images
54:     sf::Image input_image;
55:     if (!input_image.loadFromFile(input_fname)) {
56:         return -1;
57:     }
58:
59:     sf::Image output_image;
60:     if (!output_image.loadFromFile(input_fname)) {
```



```
61:         return -1;
62:     }
63:
64:     // display 2 windows
65:     sf::Vector2u imgsize = input_image.getSize();
66:     sf::RenderWindow input_window(sf::VideoMode(imgsize.x, imgsize.y), "Input Image");
67:     sf::RenderWindow output_window(sf::VideoMode(imgsize.x, imgsize.y), "Output Image");
68:
69:     // load the images into textures
70:     sf::Texture in_texture, out_texture;
71:     in_texture.loadFromImage(input_image);
72:
73:     transform(input_image, &bit_generator);
74:
75:     out_texture.loadFromImage(input_image);
76:
77:     // load textures -> sprites
78:     sf::Sprite in_sprite, out_sprite;
79:     in_sprite.setTexture(in_texture);
80:     out_sprite.setTexture(out_texture);
81:
82:     // main loop
83:     while (input_window.isOpen() && output_window.isOpen()) {
84:         sf::Event event;
85:
86:         while (input_window.pollEvent(event)) {
87:             if (event.type == sf::Event::Closed) {
88:                 input_window.close();
89:             }
90:         }
91:
92:         while (output_window.pollEvent(event)) {
93:             if (event.type == sf::Event::Closed) {
94:                 output_window.close();
95:             }
96:         }
97:
98:         input_window.clear();
99:         input_window.draw(in_sprite);    // Input image
100:        input_window.display();
101:
102:        output_window.clear();
103:        output_window.draw(out_sprite);   // Output image
104:        output_window.display();
105:    }
106:
107:    // save the image
108:    if (!input_image.saveToFile(output_fname)) {
109:        return -1;
110:    }
111:
112:    return 0;
113: }
```

```
1: /*
2: Computing IV - Assignment - PS1a + b
3: Instructor: Prof. Yelena Rykalova
4: Due Date: 02/07/22
5: Author: Matthew Lorette Anaya
6: Description: This program is an implementation of a Fibonacci Linear Feed
back          Shift Register
7:          This is a header file with the FibLFSR class definition
8: */
9:
10: #include <iostream>
11:
12: class FibLFSR {
13:
14: public:
15:     FibLFSR(std::string seed);
16:
17:     int step();
18:
19:     int generate(int k);
20:
21:     friend std::ostream& operator<<(std::ostream& os, FibLFSR &lfsr);
22:
23: private:
24:     std::string reg;
25:
26:     int getBit(char a);
27:
28:     int xOr(int a, int b);
29:
30: };
31:
32:
```

```
1: /*
2: Computing IV - Assignment - PS1a + b
3: Instructor: Prof. Yelena Rykalova
4: Due Date: 02/07/22
5: Author: Matthew Lorette Anaya
6: Description: This program is an implementation of a Fibonacci Linear Feed
back          Shift Register
7:          Takes in a seed and generates bits with seed() and numbers w
ith g         enerate(int)
8: */
9: #include <string>
10: #include <sstream>
11: #include <math.h>
12: #include "FibLFSR.h"
13:
14: FibLFSR::FibLFSR(std::string seed) {
15:     int size = seed.length();
16:     // No try-catchblock for BOOST test
17:     if(size != 16)
18:         throw std::invalid_argument("Incorect seed bit length, must be 16.");
19:     reg = seed;
20: }
21:
22: int FibLFSR::getBit(char a) {
23:     if (a == '1') return 1;
24:     else if (a == '0') return 0;
25:     else return 1;
26: }
27:
28: int FibLFSR::xOr(int a, int b) {
29:     return a != b;
30: }
31:
32: std::ostream& operator<<(std::ostream& os, FibLFSR &lfsr) {
33:     os << lfsr.reg;
34:
35:     return os;
36: }
37:
38: int FibLFSR::step() {
39:
40:     //new register after shifting
41:     std::string new_reg = reg.substr(1);
42:
43:     //Taps(10, 12, and 13)
44:     //{Equal = 0}{Not Equal = 1}
45:     int tap = xOr(reg[0], reg[2]);
46:     tap = xOr(tap, getBit(reg[3]));
47:     tap = xOr(tap, getBit(reg[5]));
48:
49:     FibLFSR::reg = new_reg;
50:     FibLFSR::reg += std::to_string(tap);
51:
52:     return tap;
53: }
54:
55: int FibLFSR::generate(int k) {
56:     int result = 0;
57:     for(int i = 0; i < k; i++){
58:         int z = step();
59:         result = (result * 2) + z;
60:     }
61:
62:     return result;
63: }
```

64:

65:

66:

```
1: // Dr. Rykalova
2: // test.cpp for PS1a
3: // updated 1/31/2020
4: /*
5: *   Computing IV - Assignment - PS1a
6: *   Instructor: Prof. Yelena Rykalova
7: *
8: *   Due Date: 01/31/22
9: *
10: *   Author: Matthew Lorette Anaya
11: *
12: *   Description: This program is an implementation of a Fibonacci Linear F
eedback Shift Register
13:
14:     Takes in a seed and generates bits with seed() and number
s with generate(int)
15:
16:     This is the test file with BOOST unit tests.
17: */
18: #include <iostream>
19: #include <string>
20:
21: #include "FibLFSR.h"
22:
23: #define BOOST_TEST_DYN_LINK
24: #define BOOST_TEST_MODULE Main
25: #include <boost/test/unit_test.hpp>
26:
27: BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
28:
29:     FibLFSR l("1011011000110110");
30:     BOOST_REQUIRE(l.step() == 0);
31:     BOOST_REQUIRE(l.step() == 0);
32:     BOOST_REQUIRE(l.step() == 0);
33:     BOOST_REQUIRE(l.step() == 1);
34:     BOOST_REQUIRE(l.step() == 1);
35:     BOOST_REQUIRE(l.step() == 0);
36:     BOOST_REQUIRE(l.step() == 0);
37:     BOOST_REQUIRE(l.step() == 1);
38:
39:     FibLFSR l2("1011011000110110");
40:     BOOST_REQUIRE(l2.generate(9) == 51);
41: }
42:
43: // Test case that prints out the starting and the resulting bit
44: // patterns whilst checking to make sure the correct result is printed
45: BOOST_AUTO_TEST_CASE(customTestCase1) {
46:     std::cout << "\n-----Custom Test Case 1-----" << std::endl;
47:     FibLFSR l("1011011000110110");
48:     std::cout << "\tOriginal seed: " << l << std::endl;
49:
50:     int res = l.generate(5);
51:     BOOST_REQUIRE(res == 3);
52:
53:     std::cout << "Results of generate(5): " << l << " " << res << std::endl
;
54:     std::cout << std::endl;
55: }
56:
57: BOOST_AUTO_TEST_CASE(customTestCase2) {
58:     std::cout << "\n-----Custom Test Case 2-----" << std::endl;
59:
60:     std::string tooShort = "10010110";
61:     std::string tooLong = "10011001001010101101";
62: }
```

```
63:  std::cout << "Test exception thrown for too short seed: 10010110" << st
d::endl;
64:  BOOST_REQUIRE_THROW(FibLFSR("10010110"), std::invalid_argument);
65:
66:  std::cout << "Test exception thrown for too long seed: 10011001001010101
101" << std::endl;
67:  BOOST_REQUIRE_THROW(FibLFSR("10011001001010101101"), std::invalid_argume
nt);
68: }
69:
```

PS2 N-Body Simulation

The purpose of this project was to mimic how pairwise forces affect particles in a universe. PS2a and PS2b were the two parts of this task. The first section of the task required you to create a representation of our galaxy using two classes: one to represent each planet and another to govern each planet. The initial part's sole purpose was to ensure that planets could be read from a text file and presented appropriately in the SFML window. The second section of the project involves simulating the effects of paired force using accurate physics calculations. The main function was required to accept data in the form of overall simulation time, time spent in one simulation step, and the input file containing information about each planet.

Key Concepts

The Universe class may be overloaded to allow for input redirection from a file to set up the planets for simulation. CelestialBody and Universe are both `sf::Drawable`, allowing them to be drawn to the window. Within the Universe class, each CelestialBody is created with a `std::unique_ptr` and stored in a `std::vector`. The planets are then rendered using the draw function on each CelestialBody in the vector in a loop in main. The majority of the calculations for this program are done in Universe's `step()` function.

What I accomplished

PS2a: Universe is a class which contains a vector of CelestialBody(s) and has a function to draw the planets by accessing a vector. For extra credit, I added a background image called "A Small Glimpse of The Cosmos:" and used smart pointers to avoid data leaks. The class Universe was a vector of CelestialBody which was used to hold all the planets that were fed into the program from planets.txt. I used a smart pointer to represent the universe, closing off the possibilities of memory issues(`unique_ptr <universe> u(new universe());`). The universe was allocated through a for loop. I had to make a pushback function in order to assign CelestialBody(s) to the vector.

PS2b: The step function was implemented in reference to the homework pdf. The step function is fed all the necessary forces via the file. Through each loop of said function, it sorts through and initializes each planet with the necessary x and y positions and velocities, acceleration, net force, force, gravitational force, and mass. Smart pointers were used to create new objects and pushback. Everything else was pretty much piggy backed off part a. I did fix my Universe class and implemented it properly. Though, looking back at it, I probably should still have kept it as its own header.

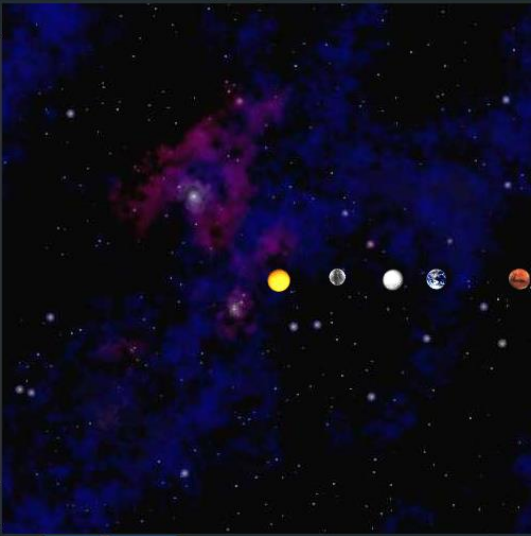
What I Learned

This task taught me a great deal. For starters, this was the first time I utilized smart pointers to initialize objects in a C++ application. I gained a lot of knowledge about the physics involved in the N-Body simulation utilizing the leapfrog finite difference approximation approach in addition to C++.

Output

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
current x Position: 3.079000e+02
current y Position: 2.500000e+02
current x Velocity: 0.000000e+00
current y Velocity: 4.790000e+04
Particle Mass: 3.302000e+23
Particle Name: mercury.gif
current x Position: 2.500000e+02
current y Position: 2.500000e+02
current x Velocity: 0.000000e+00
current y Velocity: 0.000000e+00
Particle Mass: 1.989000e+30
Particle Name: sun.gif
current x Position: 3.582000e+02
current y Position: 2.500000e+02
current x Velocity: 0.000000e+00
current y Velocity: 3.500000e+04
Particle Mass: 4.869000e+24
Particle Name: venus.gif
Loop # 0
Loop # 1
Loop # 2
Loop # 3
Loop # 4

```




```
1: C= g++
2: CFLAGS= -Wall -Werror -ansi -pedantic
3: GFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
4: c11 = -std=c++11
5:
6: all:
7:     make NBody
8:
9: NBody: CelestialBody.o main.o
10:     $(C) CelestialBody.o main.o -o NBody $(GFLAGS) $(c11)
11:
12: CelestialBody.o: CelestialBody.cpp CelestialBody.hpp
13:     $(C) -c CelestialBody.cpp -o CelestialBody.o $(CFLAGS) $(c11)
14:
15: main.o: main.cpp CelestialBody.hpp
16:     $(C) -c main.cpp -o main.o $(CFLAGS) $(c11)
17:
18: clean:
19:     rm *.o *NBody
```

```
1: #include <iostream>
2: #include <cstdlib>
3: #include <vector>
4:
5: #include <SFML/Graphics.hpp>
6: #include <SFML/Window.hpp>
7: #include <SFML/System.hpp>
8:
9:
10: #include "CelestialBody.hpp"
11:
12: using namespace std;
13:
14: int main(int argc, char* argv[]){
15:
16:     int cbodies;
17:
18:     double dt;
19:     double radius;
20:     double T;
21:     double time;
22:
23:     string filename;
24:
25:     sf::Clock clock;
26:
27:
28:     if (argc != 3){
29:
30:         cout << "\nThere are not enough arguments, exiting!" << endl;
31:         return -1;
32:     }
33:
34:     time = 0; // start time
35:
36:     filename = argv[0];
37:     T = strtod(argv[1], NULL);
38:     dt = strtod(argv[2], NULL);
39:
40:     cin >> cbodies;
41:     cin >> radius;
42:
43:
44:     Universe cb(radius, 500, cbodies, cin);
45:
46:     sf::RenderWindow window(sf::VideoMode(600, 600), "A Small Glimps of T
he Cosmos:");
47:
48:     while (window.isOpen()){
49:
50:         sf::Event event;
51:
52:         while (window.pollEvent(event)){ if (event.type == sf::Event::Clo
sed) window.close(); }
53:
54:         window.clear();
55:
56:         if (time < T){ // as long as time hasn't run out
57:
58:             sf::Time elapsed = clock.getElapsedTime();
59:
60:             cout << "\nElapsed time: " << elapsed.asSeconds( ) << " secon
ds." << endl;
61:
62:             cb.step(dt);
```

```
63:
64:         time += dt;
65:     }
66:
67:     window.draw(cb);
68:     window.display();
69: }
70:
71:     return 0;
72: }
```

```
1: #ifndef CELESTIALBODY_HPP
2: #define CELESTIALBODY_HPP
3:
4: #include <iostream>
5: #include <string>
6: #include <vector>
7: #include <memory>
8:
9: #include <SFML/Graphics.hpp>
10:
11: using namespace std;
12:
13:
14: class CelestialBody :public sf::Drawable {
15:
16:     private:
17:
18:         double winsize;
19:
20:         double xpos;
21:         double ypos;
22:
23:         double xvel;
24:         double yvel;
25:
26:         double mass;
27:         double radius;
28:
29:         double display_x;
30:         double display_y;
31:
32:         string filename;
33:
34:         sf::Sprite sprite;
35:         sf::Texture texture;
36:
37:     public:
38:         //constructors
39:         CelestialBody();
40:         CelestialBody(double x_pos, double y_pos, double x_vel, double y_vel,
double m, string name, double radius, double winsize);
41:         ~CelestialBody();
42:
43:         friend std::istream& operator >>(std::istream& input, CelestialBody&
ci);
44:         friend std::ostream& operator <<(std::ostream& out, CelestialBody& co
);
45:
46:         virtual void draw(sf::RenderTarget& target, sf::RenderStates states)c
onst;
47:
48:         //accessor functions
49:         double get_posx();
50:         double get_posy();
51:
52:         double get_velx();
53:         double get_vely();
54:
55:         double get_mass();
56:         string get_filename();
57:
58:         //mutators
59:         void set_x_y_pos(double x_input, double y_input);
60:
61:         void set_velx(double vx);
```

```
62:     void set_vely(double vy);
63:
64:     void set_radius(double radius);
65:     void set_window(double size);
66:
67:     void set_position();
68:
69: };
70:
71: class Universe : public sf::Drawable {
72:
73:     public:
74:
75:     Universe(); // basic constructor
76:     Universe(double radius, int window, int num_of_planets, istream &in);
77:
78:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
79:
80:     friend ostream &operator <<(std::ostream& out, const Universe& co);
81:
82:     void step(double seconds);
83:
84:     double get_r();
85:     int get_numPlanets();
86:
87:     void printInfo();
88:
89:     private:
90:
91:     double r;
92:
93:     int numplanets;
94:     int winsize;
95:
96:     vector <std::unique_ptr <CelestialBody>> planets;
97:
98: };
99:
100: #endif
```

```
1: #include "CelestialBody.hpp"
2:
3: #include <iostream>
4: #include <cmath>
5:
6:
7: using namespace std;
8:
9:
10: Universe::Universe() {
11:
12:     r = 0;
13:     winsize = 0;
14: }
15:
16: Universe::Universe(double radius, int window, int num_of_planets, istream
&in) {
17:
18:     int i;
19:
20:     r = radius;
21:     winsize = window;
22:     numplanets = num_of_planets;
23:
24:     for (i = 0; i < num_of_planets; i++) {
25:
26:         unique_ptr <CelestialBody> ptr(new CelestialBody());
27:
28:         CelestialBody();
29:         planets.push_back(move(ptr));
30:         planets[i]->set_radius(r);
31:         planets[i]->set_window(window);
32:         in >> *planets[i];
33:     }
34:
35: }
36:
37: void Universe::draw(sf::RenderTarget &target, sf::RenderStates states) co
nst {
38:
39:     int i;
40:
41:     for (i = 0; i < numplanets; i++) {
42:         target.draw(*planets.at(i), states);
43:     }
44: }
45:
46:
47: void Universe::step(double seconds) {
48:
49:     int i, k;
50:
51:     double ax;
52:     double ay;
53:
54:     double dx;
55:     double dy;
56:
57:     double force;
58:     double forcex;
59:     double forcey;
60:
61:     double fx;
62:     double fy;
63:
```

```
64:     double G;
65:
66:     double velx;
67:     double vely;
68:
69:     double x2;
70:     double y2;
71:
72:     double r;
73:
74:     for (i = 0; i < numplanets; i++){
75:         fx = 0;
76:         fy = 0;
77:
78:         for (k = 0; k < numplanets; k++){
79:
80:             if (k != i){
81:
82:                 G = 6.67e-11; // gravitational constant
83:
84:                 dx = planets[k]->get_posx() - planets[i]->get_posx();
85:                 dy = planets[k]->get_posy() - planets[i]->get_posy();
86:
87:                 r = sqrt(pow(dx, 2) + pow(dy, 2));
88:
89:                 force = (G * planets[k]->get_mass() * planets[i]->get_mass()) / po
w(r, 2);
90:                 forcex = force * (dx / r);
91:                 forcey = force * (dy / r);
92:
93:                 fy += forcey;
94:                 fx += forcex;
95:
96:             }
97:         }
98:
99:         ax = fx / planets[i]->get_mass();
100:         ay = fy / planets[i]->get_mass();
101:
102:         velx = planets[i]->get_velx() + seconds * ax;
103:         vely = planets[i]->get_vely() + seconds * ay;
104:
105:         planets[i]->set_velx(velx);
106:         planets[i]->set_vely(vely);
107:
108:         x2 = (planets[i]->get_posx()) + velx * seconds;
109:         y2 = (planets[i]->get_posy()) + vely * seconds;
110:
111:         planets[i]->set_x_y_pos(x2, y2);
112:     }
113: }
114:
115: void Universe::printInfo(){
116:
117:     int i; // for loop
118:
119:     cout << numplanets << endl;
120:     cout << r << endl;
121:     for (i = 0; i < numplanets; i++){
122:         cout << planets[i]->get_posx() << " " << planets[i]->get_posy() << "
"
123:         << planets[i]->get_velx() << " " << planets[i]->get_vely() << " "
124:         << planets[i]->get_mass() << " " << planets[i]->get_filename() << e
ndl;
125:     } // print out x pos, y pos, velx, vely, mass, and name
```

```
126: }
127:
128: double Universe::get_r(){ return r; }
129:
130: int Universe::get_numPlanets(){ return numplanets; }
131:
132: CelestialBody::CelestialBody(){
133:
134:     winsize = 0;
135:     xpos = 0;
136:     ypos = 0;
137:     xvel = 0;
138:     yvel = 0;
139:     mass = 0;
140:     radius = 0;
141:     filename = "";
142: }
143:
144: CelestialBody::CelestialBody(double x_pos, double y_pos, double x_vel,
145:                               double y_vel, double m, string name,
146:                               double rad, double window_size){
147:
148:     double radx;
149:     double rady;
150:
151:     xpos = x_pos; // updated values of xpos
152:     ypos = y_pos; // updated values of ypos
153:     xvel = x_vel; // updated values of xvel
154:     yvel = y_vel; // updated values of yvel
155:
156:     mass = m; // update mass
157:     radius = rad; // update radius
158:     winsize = window_size; // update window size
159:     filename = name; // update filename
160:
161:     radx = (winsize / 2) * (xpos / radius) + (winsize / 2);
162:     rady = (winsize / 2) * (ypos / radius) + (winsize / 2);
163:
164:
165:     texture.loadFromFile(filename);
166:
167:
168:     sprite.setTexture(texture);
169:     sprite.setPosition(radx, rady);
170: }
171:
172: istream &operator >>(istream &in, CelestialBody &ci){
173:
174:     double radx;
175:     double rady;
176:
177:     in >> ci.xpos >> ci.ypos >> ci.xvel >> ci.yvel >> ci.mass >> ci.filename
e;
178:
179:     radx = (ci.winsize / 2) * (ci.xpos / ci.radius) + (ci.winsize / 2);
180:     rady = (ci.winsize / 2) * (ci.ypos / ci.radius) + (ci.winsize / 2);
181:
182:     ci.texture.loadFromFile(ci.filename);
183:
184:     ci.sprite.setTexture(ci.texture);
185:     ci.sprite.setPosition(radx, rady);
186:
187:
188:     return in; // return input
189: }
```



```
190:
191: void CelestialBody::draw(sf::RenderTarget &target, sf::RenderStates state
s) const { target.draw(sprite, states); }
192:
193: CelestialBody::~CelestialBody(){}
194:
195: double CelestialBody::get_posx(){ return xpos; } // return the xpos
196: double CelestialBody::get_posy(){ return ypos; } // return the ypos
197: double CelestialBody::get_velx(){ return xvel; } // return the xvel
198: double CelestialBody::get_vely(){ return yvel; } // return the yvel
199: double CelestialBody::get_mass(){ return mass; } // return the mass
200:
201: string CelestialBody::get_filename(){ return filename; }
202:
203: void CelestialBody::set_radius(double r){ radius = r; }
204: void CelestialBody::set_window(double size){ winsize = size; }
205: void CelestialBody::set_velx(double vx){ xvel = vx; }
206: void CelestialBody::set_vely(double vy){ yvel = vy; }
207: void CelestialBody::set_x_y_pos(double x_input, double y_input){
208:
209:     double radx;
210:     double rady;
211:
212:     xpos = x_input;
213:     ypos = y_input;
214:
215:     radx = (winsize / 2) * (xpos / radius) + (winsize / 2);
216:     rady = (winsize / 2) * (-ypos / radius) + (winsize / 2);
217:
218:     sprite.setPosition(sf::Vector2f(radx, rady));
219: }
```

PS3 Recursive Graphics (Triangle Fractal)

For this assignment, we were given the task of implementing the Sierpinski triangle project from Princeton. The assignment's main objective was to utilize recursion to create a complex-looking triangle using only a few lines of code. The main program took an integer, N, and used it to change the depth of the recursion. After that, our software would recursively draw triangles within triangles, drawing one triangle at depth 1, four triangles at depth 2, and so on.

Key concepts

Recursion is literally the name of the game here within this project. It allows the triangles to be printed to pascals form without manual input or more tedious loops. Vectors are also a godsend when it comes to memory management, which can be a nightmare if the changing of an objects "size" is needed or leaks are allowed to happen.

What I Accomplished

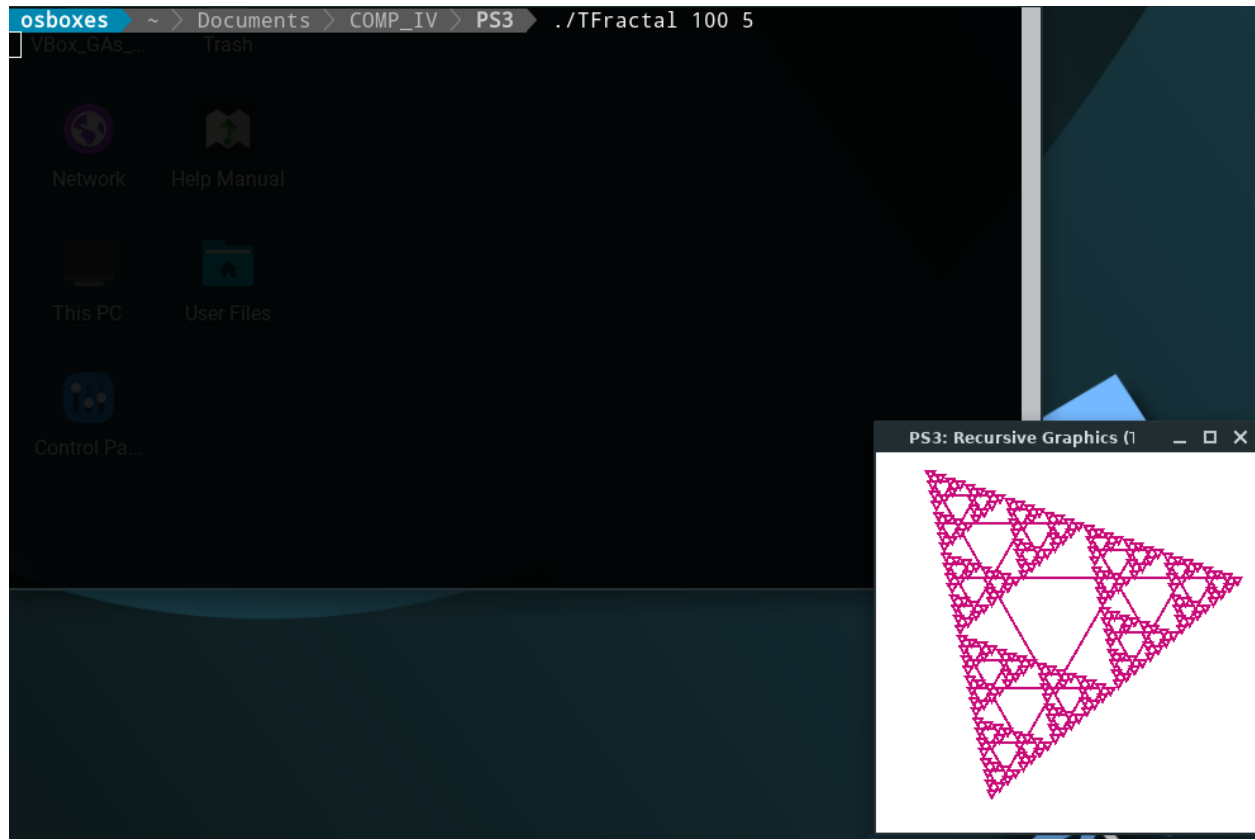
The project contains an object that holds all the information for each individual triangle. Main.cpp then uses a recursive function to print them All out in a pascal triangle. This function is pretty much the whole project. It sets the size, thickness, and color of the triangles to be printed by main.

```
Triangle::Triangle(double initX, double initY, double initL):
    x(initX), y(initY), l(initL) {
    double i = sqrt(.75 * pow(l, 2));
    triangle.setPointCount(3);
    triangle.setPoint(0, Vector2f(static_cast<float>(0),
        static_cast<float>(0)));
    triangle.setPoint(1, Vector2f(static_cast<float>(l),
        static_cast<float>(0)));
    triangle.setPoint(2, Vector2f(static_cast<float>(l/2),
        static_cast<float>(i)));
    sf::Color color((time(0) * 5 + offset1) % 256,
        (time(0) * 5 + offset2) % 256, (time(0) * 5 + offset3) % 256);
    triangle.setOutlineColor(color);
    triangle.setOutlineThickness(2);
    triangle.setPosition(x, y);
```

What I Learned

That recursion is a really powerful tool, but can be extremely taxing if not implemented properly. The memory management can easily get out of hand with a slight miscalculation

Output



```
1: C= g++
2: CFLAGS= -Wall -Werror -ansi -pedantic
3: GFLAGS= -lsfml-graphics -lsfml-window -lsfml-system
4: c17 = -std=c++17
5:
6: all: TFractal
7:
8: TFractal: TFractal.o Triangle.o
9:      $(C) TFractal.o Triangle.o -o TFractal $(GFLAGS)
10:
11: TFractal.o: TFractal.cpp
12:      $(C) -c TFractal.cpp -o TFractal.o $(CFLAGS) $(c17)
13:
14: Triangle.o: Triangle.cpp Triangle.hpp
15:      $(C) -c Triangle.cpp -o Triangle.o $(CFLAGS) $(c17)
16:
17: clean:
18:      rm *.o *TFractal
```

```
1: #include <iostream>
2: #include <SFML/Graphics.hpp>
3: #include "Triangle.hpp"
4:
5: using std::cout;
6: using std::endl;
7: using std::stod;
8: using sf::RenderWindow;
9: using std::stoi;
10:
11: void triangleFractal(int i, RenderWindow* window,
12:     double x, double y, double l) {
13:
14:     Triangle triangle(x, y, l);
15:     window->draw(triangle);
16:     if (i > 0) {
17:         triangleFractal(i - 1, window, x - (l / 4),
18:             y - sqrt((3.0/16) * pow(l, 2)), l/2);
19:         triangleFractal(i - 1, window, x + l, y, l/2);
20:         triangleFractal(i - 1, window, x, y + sqrt(.75 * pow(l, 2)), l/2)
;
21:     }
22:     return;
23: }
24:
25: int main(int argc, char* argv[]) {
26:     // sets up command line arguments
27:     if (argc != 3) {
28:         cout << "Incorrect number of inputs." << endl;
29:         exit(1);
30:     }
31:     int N = stoi(argv[2]);
32:     double L = stod(argv[1]);
33:     if (L <= 0 || N <= 0) {
34:         cout << "Incorrect input range." << endl;
35:     }
36:
37:     RenderWindow window(sf::VideoMode(L * 3, L * 3), "PS3: Recursive Grap
hics (Triangle Fractal)");
38:     // loop to check if closed
39:     while (window.isOpen()) {
40:         sf::Event event;
41:         while (window.pollEvent(event)) {
42:             if (event.type == sf::Event::Closed)
43:                 window.close();
44:         }
45:
46:         // fractal triangle setup
47:         window.clear(sf::Color::White);
48:         triangleFractal(N, &window, L * (9.0/10), L, L);
49:
50:         window.display();
51:     }
52:     return 0;
53: }
```

```
1: #ifndef TRIANGLE_HPP
2: #define TRIANGLE_HPP
3:
4: #include <cmath>
5: #include <SFML/Graphics.hpp>
6:
7: using sf::ConvexShape;
8: using sf::Vector2f;
9:
10: class Triangle : public sf::Drawable{
11: public:
12:     Triangle(double initX, double initY, double initL);
13: private:
14:     ConvexShape triangle;
15:     double x, y;
16:     double l;
17:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
const;
18: };
19:
20: #endif
```

```
1: #include "Triangle.hpp"
2: #include <ctime>
3: #include <random>
4:
5: std::random_device rd;
6: std::mt19937 mt(rd());
7: std::uniform_int_distribution<int> dist(0, 255);
8:
9: int offset1 = dist(mt);
10: int offset2 = dist(mt);
11: int offset3 = dist(mt);
12:
13: void Triangle::draw(sf::RenderTarget& target, sf::RenderStates states) co
nst {
14:     target.draw(triangle, states);
15: }
16:
17: Triangle::Triangle(double initX, double initY, double initL):
18:     x(initX), y(initY), l(initL) {
19:     double i = sqrt(.75 * pow(l, 2));
20:     triangle.setPointCount(3);
21:     triangle.setPoint(0, Vector2f(static_cast<float>(0),
22:         static_cast<float>(0)));
23:     triangle.setPoint(1, Vector2f(static_cast<float>(1),
24:         static_cast<float>(0)));
25:     triangle.setPoint(2, Vector2f(static_cast<float>(1/2),
26:         static_cast<float>(i)));
27:     sf::Color color((time(0) * 5 + offset1) % 256,
28:         (time(0) * 5 + offset2) % 256, (time(0) * 5 + offset3) % 256);
29:     triangle.setOutlineColor(color);
30:     triangle.setOutlineThickness(2);
31:     triangle.setPosition(x, y);
32: }
```

PS4 Synthesizing a Plucked String Sound

The Karplus-Strong algorithm was used to simulate a plucked guitar string sound in this project. PS4a and PS4b were the two portions of the assignment. In PS4a, we were supposed to create the CircularBuffer class, which would be utilized to keep a ring buffer feedback mechanism running. PS4b would then implement the StringSound class, which would create a CircularBuffer with N samples depending on a 44,100Hz sample rate and a set frequency. The primary function's key pushes would be paired with these frequencies, which would be broadcast via SFML audio.

Key concepts

Due to the fact that StringSound uses CircularBuffer as a dynamic array, each class has to deal with dynamic memory allocation and copy/move/destructor operations. Exceptions were utilized for various functions in both classes. The StringSound(frequency) constructor, for example, includes one exception that precludes it from being used with a frequency of 0. When dynamically allocating memory for the buffer during construction and utilizing the CircularBuffer routines in its own functions, StringSound handles errors given by CircularBuffer.

What I Accomplished

PS4a: CircularBuffer simulates a ring buffer feedback mechanism by storing N vibration samples in an array of type `int16_t`. To operate with the Karplus-Strong algorithm, the class implements the array as a queue, containing operations like enqueue and dequeue. Exception handling is included in the CircularBuffer constructor and queue routines, which I tested using Boost libraries in test.cpp.

PS4b: StringSound builds a CircularBuffer of length equal to sample rate / frequency from a frequency. To mimic noise, the function `pluck()` fills the buffer of a string sound with random values in the `Int16` range. The sound's buffer is advanced to the next stage by the function `tic()`, and the `sample()` function returns the first sample from the buffer. The main method first constructs a vector of `Int16` vectors containing the samples of each string sound, which is then loaded into a vector of `sf::SoundBuffers`, which is then loaded into a vector of `sf::Sounds`. A sound will be played inside the SFML display loop when the relevant key is hit.

What I Learned

I gained a lot of knowledge about the Karplus-Strong algorithm for streaming digital music and how to interact with audio in SFML. Speaking of which, SFML's Keyboard library is quite useful for handling a piano, guitar, or other instrument – I was thinking of how it might be used in basic 2D games, which, when paired with the PS2's planet things, could produce a very good space invaders-style game.


```
1: CC= g++
2: CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
3: SFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
4: BOOST= -lboost_unit_test_framework
5:
6:
7: all: KSGuitarSim SStest SSLite
8:
9: KSGuitarSim: KSGuitarSim.o StringSound.o CircularBuffer.o
10:      $(CC) KSGuitarSim.o StringSound.o CircularBuffer.o -o KSGuitarSim
$(SFLAGS)
11:
12: SSLite: SSLite.o StringSound.o CircularBuffer.o
13:      $(CC) SSLite.o StringSound.o CircularBuffer.o -o SSLite $(SFLAGS)
14:
15: SStest: SStest.o StringSound.o CircularBuffer.o
16:      $(CC) SStest.o StringSound.o CircularBuffer.o -o SStest $(BOOST)
17:
18: KSGuitarSim.o: KSGuitarSim.cpp StringSound.hpp
19:      $(CC) -c KSGuitarSim.cpp StringSound.hpp $(CFLAGS)
20:
21: SSLite.o: SSLite.cpp StringSound.hpp
22:      $(CC) -c SSLite.cpp StringSound.hpp $(CFLAGS)
23:
24: StringSound.o: StringSound.cpp StringSound.hpp
25:      $(CC) -c StringSound.cpp StringSound.hpp $(CFLAGS)
26:
27: CircularBuffer.o: CircularBuffer.cpp CircularBuffer.hpp
28:      $(CC) -c CircularBuffer.cpp CircularBuffer.hpp $(CFLAGS)
29:
30: SStest.o: SStest.cpp
31:      $(CC) -c SStest.cpp $(Boost)
32:
33: clean:
34:      rm *.o
35:      rm *.gch
36:      rm KSGuitarSim
37:      rm SStest
38:      rm SSLite
39:
```

```
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.hpp"
16: #include "StringSound.hpp"
17:
18: #define CONCERT_A 440.0
19: #define SAMPLES_PER_SEC 44100
20: const int keyboard_size = 37;
21:
22: std::vector<sf::Int16> makeSamples(StringSound gs)
23: {
24:     std::vector<sf::Int16> samples;
25:
26:     gs.pluck();
27:     int duration = 8; // seconds
28:     int i;
29:     for (i = 0; i < SAMPLES_PER_SEC * duration; i++) {
30:         gs.tic();
31:         samples.push_back(gs.sample());
32:     }
33:
34:     return samples;
35: }
36:
37: int main()
38: {
39:     sf::RenderWindow window(sf::VideoMode(800, 800), "SFML KSGuitarSim");
40:     sf::Event event;
41:
42:     double frequency;
43:     std::vector<sf::Int16> sample;
44:
45:     std::vector<std::vector<sf::Int16>> samples(keyboard_size);
46:     std::vector<sf::SoundBuffer> buffers(keyboard_size);
47:     std::vector<sf::Sound> sounds(keyboard_size);
48:
49:     std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
50:
51:     for (int i = 0; i < (signed)keyboard.size(); i++) {
52:         frequency = CONCERT_A * pow(2, ((i - 24) / 12.0));
53:         StringSound tmp = StringSound(frequency);
54:
55:         sample = makeSamples(tmp);
56:         samples[i] = sample;
57:
58:         if (!buffers[i].loadFromSamples(&samples[i][0],
59:                                     samples[i].size(), 2, SAMPLES_PER
60: _SEC)) {
61:             throw std::runtime_error("sf::SoundBuffer: failed to load fro
62 m samples.");
63:         }
64:         sounds[i].setBuffer(buffers[i]);
65:     }
66: }
```

```
64:     }
65:
66:     while (window.isOpen()) {
67:         while (window.pollEvent(event)) {
68:             if (event.type == sf::Event::TextEntered) {
69:                 if (event.text.unicode < 128) {
70:                     char key = static_cast<char>(event.text.unicode);
71:
72:                     for (int i = 0; i < (signed)keyboard.size(); i++) {
73:                         if (keyboard[i] == key) {
74:                             std::cout << "Keyboard key is: " << keyboard[
i] << "\n";
75:                             std::cout << "Attempting to play sound...\n";
76:                             sounds[i].play();
77:                             break;
78:                         }
79:                     }
80:                 }
81:             }
82:
83:             if (event.type == sf::Event::Closed) {
84:                 window.close();
85:             }
86:         }
87:
88:         window.clear();
89:         window.display();
90:     }
91:     return 0;
92: }
```

```
1: // Copyright 2022 Matthew Lorette Anaya, matthew_loretteanaya@student.uml
.edu
2:
3: #ifndef _USERS_MATTHEWLORETTEANAYA_DOCUMENTS_UML_COMP_IV_PS4B_CIRCULARBUF
FER_HPP_
4: #define _USERS_MATTHEWLORETTEANAYA_DOCUMENTS_UML_COMP_IV_PS4B_CIRCULARBUF
FER_HPP_
5:
6: #include <stdint.h>
7: #include <iostream>
8: #include <string>
9: #include <sstream>
10: #include <exception>
11: #include <stdexcept>
12: #include <vector>
13:
14: class CircularBuffer {
15: public:
16: // create an empty circular buffer, with given max capacity
17: explicit CircularBuffer(int capacity);
18: int size();
19: bool isEmpty();
20: bool isFull();
21: void enqueue(int16_t x);
22: int16_t dequeue();
23: int16_t peek();
24: void output();
25:
26: private:
27: std::vector<int16_t> buff;
28: int first;
29: int last;
30: int cap;
31: int s;
32: };
33:
34: #endif // _USERS_MATTHEWLORETTEANAYA_DOCUMENTS_UML_COMP_IV_PS4B_CIRCULAR
BUFFER_HPP_
```

```
1:  /*
2:   Copyright 2022 Matthew Lorette Anaya, matthew_loretteanaya@student.uml.
edu
3:  */
4:
5:  #include "CircularBuffer.hpp"
6:
7:  CircularBuffer::CircularBuffer(int capacity) {
8:      if (capacity < 1) {
9:          throw std::invalid_argument
10:             ("Circular Buffer constructor: capacity must be greater than zer
o");
11:      }
12:
13:      last = 0;
14:      first = 0;
15:      s = 0;
16:      cap = capacity;
17:      buff.resize(capacity);
18:
19:      return;
20: }
21:
22: int CircularBuffer::size() {
23:     return s;
24: }
25:
26: bool CircularBuffer::isEmpty() {
27:     if (s != 0) {
28:         return false;
29:     } else {
30:         return true;
31:     }
32: }
33:
34: bool CircularBuffer::isFull() {
35:     if (s == cap) {
36:         return true;
37:     } else {
38:         return false;
39:     }
40: }
41: void CircularBuffer::enqueue(int16_t x) {
42:     if (isFull()) {
43:         throw std::runtime_error("enqueue: can't enqueue to a full buffer
");
44:     }
45:     if (last >= cap) {
46:         last = 0;
47:     }
48:     // Continue
49:     buff.at(last) = x;
50:     last++;
51:     s++;
52: }
53:
54: int16_t CircularBuffer::dequeue() {
55:     if (isEmpty()) {
56:         throw std::runtime_error("dequeue: can't dequeue an empty buffer"
);
57:     }
58:     int16_t retFirst = buff.at(first);
59:     buff.at(first) = 0;
60:     first++;
61:     s--;
```

```
62:
63:     if (first >= cap) {
64:         first = 0;
65:     }
66:
67:     return retFirst;
68: }
69:
70: int16_t CircularBuffer::peek() {
71:     if (isEmpty()) {
72:         throw std::runtime_error("peek: can't peek an empty buffer");
73:     }
74:     return buff.at(first);
75: }
76:
77: void CircularBuffer::output() {
78:     std::cout << "    First: " << first << "\n";
79:     std::cout << "    Last: " << last << "\n";
80:     std::cout << "Capacity: " << cap << "\n";
81:     std::cout << "    Size: " << s << "\n";
82:     std::cout << "Vector size: " << buff.size() << "\n";
83:     std::cout << "Vector capacity: " << buff.capacity() << "\n";
84:     std::cout << "Buffer (no blanks): \n";
85:
86:     int x = 0;
87:     int y = first;
88:
89:     while (x < s) {
90:         // Make the loop go back to 0 to continue printing.
91:         if (y >= cap) {
92:             y = 0;
93:         }
94:
95:         std::cout << buff[y] << " ";
96:         y++;
97:         x++;
98:     }
99:
100:     std::cout << "\nDump the entire buffer (including blanks): \n";
101:
102:     for (int x = 0; x < cap; x++) {
103:         std::cout << buff[x] << " ";
104:     }
105:
106:     std::cout << "\n\n";
107: }
```

```
1: #ifndef STRINGSOUND_HPP
2: #define STRINGSOUND_HPP
3:
4: #include <SFML/Audio.hpp>
5: #include <SFML/Graphics.hpp>
6: #include <SFML/System.hpp>
7: #include <SFML/Window.hpp>
8: #include <cmath>
9: #include <iostream>
10: #include <string>
11: #include <vector>
12: #include "CircularBuffer.hpp"
13:
14: const int SAMPLING_RATE = 44100;
15: const double ENERGY_DECAY_FACTOR = 0.996;
16:
17: class StringSound {
18: public:
19:     explicit StringSound(double frequency);
20:
21:     explicit StringSound(std::vector<sf::Int16> init);
22:
23:     void pluck();
24:
25:     // advance the simulation one time step
26:     void tic();
27:
28:     // return the current sample
29:     sf::Int16 sample();
30:
31:     // return number of times tic was called
32:     int time();
33:
34: private:
35:     CircularBuffer buff;
36:     int num;
37:     int tictic;
38: };
39: #endif
```

```
1: #include "StringSound.hpp"
2: #include <vector>
3:
4:
5: StringSound::StringSound(double frequency):
6:     buff(ceil(SAMPLING_RATE / frequency)) {
7:     num = ceil(SAMPLING_RATE / frequency);
8:
9:     for (int i = 0; i < num; i++) {
10:         buff.enqueue((int16_t)0);
11:     }
12:     tictic = 0;
13: }
14:
15:
16: StringSound::StringSound(std::vector<sf::Int16> init):
17:     buff(init.size()) {
18:     num = init.size();
19:
20:     std::vector<sf::Int16>::iterator it;
21:
22:     for (it = init.begin(); it < init.end(); it++) {
23:         buff.enqueue((int16_t)*it);
24:     }
25:     tictic = 0;
26: }
27:
28: void StringSound::pluck() {
29:     for (int i = 0; i < num; i++) {
30:         buff.dequeue();
31:     }
32:
33:     for (int i = 0; i < num; i++) {
34:         buff.enqueue((sf::Int16)(rand() & 0xffff));
35:     }
36:
37:     return;
38: }
39:
40:
41: void StringSound::tic() {
42:     int16_t first = buff.dequeue();
43:     int16_t second = buff.peek();
44:
45:     int16_t avg = (first + second) / 2;
46:     int16_t karplus = avg * ENERGY_DECAY_FACTOR;
47:
48:     buff.enqueue((sf::Int16)karplus);
49:
50:     tictic++;
51:
52:     return;
53: }
54:
55:
56: // return current sample
57: sf::Int16 StringSound::sample() {
58:
59:     sf::Int16 sample = (sf::Int16)buff.peek();
60:
61:     return sample;
62: }
63:
64:
65: // number of tics called
```



```
66: int StringSound::time() {  
67:     return tictic;  
68: }
```

```
1:  /*
2:   Copyright 2015 Fred Martin,
3:   Y. Rykalova, 2020
4:  */
5:
6:  #include <SFML/Graphics.hpp>
7:  #include <SFML/System.hpp>
8:  #include <SFML/Audio.hpp>
9:  #include <SFML/Window.hpp>
10:
11:  #include <math.h>
12:  #include <limits.h>
13:
14:  #include <iostream>
15:  #include <string>
16:  #include <exception>
17:  #include <stdexcept>
18:  #include <vector>
19:
20:  #include "CircularBuffer.hpp"
21:  #include "StringSound.hpp"
22:
23:  using namespace std;
24:
25:  #define CONCERT_A 220.0
26:  #define SAMPLES_PER_SEC 44100
27:
28:  vector<sf::Int16> makeSamples(StringSound gs) {
29:      std::vector<sf::Int16> samples;
30:
31:      gs.pluck();
32:      int duration = 8; // seconds
33:      int i;
34:      for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
35:          gs.tic();
36:          samples.push_back(gs.sample());
37:      }
38:
39:      return samples;
40:  }
41:
42:  int main() {
43:      sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Plucked String S
ound Lite");
44:      sf::Event event;
45:      double freq;
46:      vector<sf::Int16> samples;
47:      freq = CONCERT_A;
48:      StringSound gs1 = StringSound(freq);
49:      sf::Sound sound1;
50:      sf::SoundBuffer buf1;
51:      samples = makeSamples(gs1);
52:      if (!buf1.loadFromSamples(&samples[0], samples.size(), 2, SAMPLES_PER_S
EC))
53:          throw std::runtime_error("sf::SoundBuffer: failed to load from samp
les.");
54:      sound1.setBuffer(buf1);
55:
56:      freq = CONCERT_A * pow(2, 3.0/12.0);
57:      StringSound gs2 = StringSound(freq);
58:      sf::Sound sound2;
59:      sf::SoundBuffer buf2;
60:      samples = makeSamples(gs2);
61:      if (!buf2.loadFromSamples(&samples[0], samples.size(), 2, SAMPLES_PER_S
EC))
```

```
62:         throw std::runtime_error("sf::SoundBuffer: failed to load from samp
les.");
63:     sound2.setBuffer(buf2);
64:
65:     while (window.isOpen()) {
66:         while (window.pollEvent(event)) {
67:             switch (event.type) {
68:                 case sf::Event::Closed:
69:                     window.close();
70:                     break;
71:
72:                 case sf::Event::KeyPressed:
73:                     switch (event.key.code) {
74:                         case sf::Keyboard::A:
75:                             sound1.play();
76:                             break;
77:                         case sf::Keyboard::C:
78:                             sound2.play();
79:                             break;
80:                         default:
81:                             break;
82:                     }
83:
84:                 default:
85:                     break;
86:             }
87:
88:             window.clear();
89:             window.display();
90:         }
91:     }
92:     return 0;
93: }
94:
```

```
1: #define BOOST_TEST_DYN_LINK
2: #define BOOST_TEST_MODULE Main
3: #include <boost/test/unit_test.hpp>
4:
5: #include <vector>
6: #include <exception>
7: #include <stdexcept>
8:
9: #include "StringSound.hpp"
10:
11: BOOST_AUTO_TEST_CASE(GS)
12: {
13:     std::vector<sf::Int16> v;
14:     v.push_back(0);
15:     v.push_back(2000);
16:     v.push_back(4000);
17:     v.push_back(-10000);
18:
19:     BOOST_REQUIRE_NO_THROW(StringSound ss = StringSound(v));
20:
21:     StringSound ss = StringSound(v);
22:
23:     // StringSound = 0 2000 4000 -10000
24:     BOOST_REQUIRE(ss.sample() == 0);
25:
26:     // StringSound = 2000 4000 -10000 996
27:     ss.tic();
28:     BOOST_REQUIRE(ss.sample() == 2000);
29:
30:     // StringSound = 4000 -10000 996 2988
31:     ss.tic();
32:     BOOST_REQUIRE(ss.sample() == 4000);
33:
34:     // StringSound = -10000 996 2988 -2988
35:     ss.tic();
36:     BOOST_REQUIRE(ss.sample() == -10000);
37:
38:     // StringSound = 996 2988 -2988 -4483
39:     ss.tic();
40:     BOOST_REQUIRE(ss.sample() == 996);
41:
42:     // StringSound = 2988 -2988 -4483 1984
43:     ss.tic();
44:     BOOST_REQUIRE(ss.sample() == 2988);
45:
46:     // StringSound = -2988 -4483 1984 0
47:     ss.tic();
48:     BOOST_REQUIRE(ss.sample() == -2988);
49:
50:     //more
51:     ss.tic();
52:     BOOST_REQUIRE(ss.sample() == -4483);
53:     ss.tic();
54:     BOOST_REQUIRE(ss.sample() == 1984);
55:     ss.tic();
56:     BOOST_REQUIRE(ss.sample() == 0);
57: }
```

```
1: #include "CircularBuffer.hpp"
2:
3: int main(){
4:     std::cout << "Test main.\n";
5:
6:     CircularBuffer test(100);
7:     test.enqueue(1);
8:     test.enqueue(2);
9:     test.enqueue(3);
10:    std::cout << "Peek: " << test.peek() << "\n";
11:
12:    std::cout << "Deq 1: " << test.dequeue() << "\n";
13:    std::cout << "Deq 2: " << test.dequeue() << "\n";
14:
15:    test.output();
16:
17:    // Test looping back around
18:    CircularBuffer test2(3);
19:
20:    test2.enqueue(1);
21:    test2.enqueue(2);
22:    test2.enqueue(3);
23:
24:    test2.dequeue();
25:    test2.dequeue();
26:    test2.dequeue();
27:
28:    test2.enqueue(1);
29:    test2.enqueue(2);
30:    test2.enqueue(3);
31:    test2.dequeue();
32:    test2.enqueue(4);
33:
34:    test2.output();
35:
36:    return 0;
37: }
```

```
1: #define BOOST_TEST_DYN_LINK
2: #define BOOST_TEST_MODULE Main
3: #include <boost/test/unit_test.hpp>
4:
5: #include "CircularBuffer.hpp"
6:
7: // Tests various aspects of the constructor.
8: BOOST_AUTO_TEST_CASE(Constructor)
9: {
10:     // Shouldn't fail.
11:     BOOST_REQUIRE_NO_THROW(CircularBuffer(100));
12:
13:     // Should fail.
14:     BOOST_REQUIRE_THROW(CircularBuffer(0), std::exception);
15:     BOOST_REQUIRE_THROW(CircularBuffer(0), std::invalid_argument);
16:     BOOST_REQUIRE_THROW(CircularBuffer(-1), std::invalid_argument);
17: }
18:
19: // Checks the size() method
20: BOOST_AUTO_TEST_CASE(Size)
21: {
22:     CircularBuffer test(1);
23:
24:
25:     BOOST_REQUIRE(test.size() == 0);
26:
27:     test.enqueue(5);
28:
29:
30:     BOOST_REQUIRE(test.size() == 1);
31:
32:     test.dequeue();
33:     BOOST_REQUIRE(test.size() == 0);
34: }
35:
36: // Checks the isEmpty() method
37: BOOST_AUTO_TEST_CASE(isEmpty)
38: {
39:     // True
40:     CircularBuffer test(5);
41:     BOOST_REQUIRE(test.isEmpty() == true);
42:
43:     // False
44:     CircularBuffer test2(5);
45:     test2.enqueue(5);
46:     BOOST_REQUIRE(test2.isEmpty() == false);
47: }
48:
49: // Checks the isFull() method
50: BOOST_AUTO_TEST_CASE(isFull)
51: {
52:     CircularBuffer test(5);
53:     BOOST_REQUIRE(test.isFull() == false);
54:
55:     CircularBuffer test2(1);
56:     test2.enqueue(5);
57:     BOOST_REQUIRE(test2.isFull() == true);
58: }
59:
60: // Test enqueue
61: BOOST_AUTO_TEST_CASE(Enqueue)
62: {
63:     // These test basic enqueueing
64:     CircularBuffer test(5);
65:
```

```
66: BOOST_REQUIRE_NO_THROW(test.enqueue(1));
67: BOOST_REQUIRE_NO_THROW(test.enqueue(2));
68: BOOST_REQUIRE_NO_THROW(test.enqueue(3));
69: BOOST_REQUIRE_NO_THROW(test.enqueue(4));
70: BOOST_REQUIRE_NO_THROW(test.enqueue(5));
71: BOOST_REQUIRE_THROW(test.enqueue(6), std::runtime_error);
72: }
73:
74: // Test dequeue
75: BOOST_AUTO_TEST_CASE(Dequeue)
76: {
77:     CircularBuffer test(5);
78:
79:     test.enqueue(0);
80:     test.enqueue(1);
81:     test.enqueue(2);
82:
83:     BOOST_REQUIRE(test.dequeue() == 0);
84:     BOOST_REQUIRE(test.dequeue() == 1);
85:     BOOST_REQUIRE(test.dequeue() == 2);
86:     BOOST_REQUIRE_THROW(test.dequeue(), std::runtime_error);
87: }
```

PS5 DNA Sequence Alignment

This task required us to compare two ASCII strings in order to determine their edit distance, as well as provide space and time analysis on the program while it was running. Using a dynamic programming method, we were required to create a program that computed an optimum sequence alignment on two DNA sequences. The space complexity of our program might then be measured using Valgrind, a programming tool.

Key concepts

The EDistance class represents a matrix of integers with a vector of vectors of type int. The function optDistance() would then use the min() and penalty() functions to fill the matrix from bottom to top with the smallest of three distances, then return the optimal distance between the two sequences. The alignment() method would then go backwards through the populated matrix, constructing the output alignment string depending on the best path.

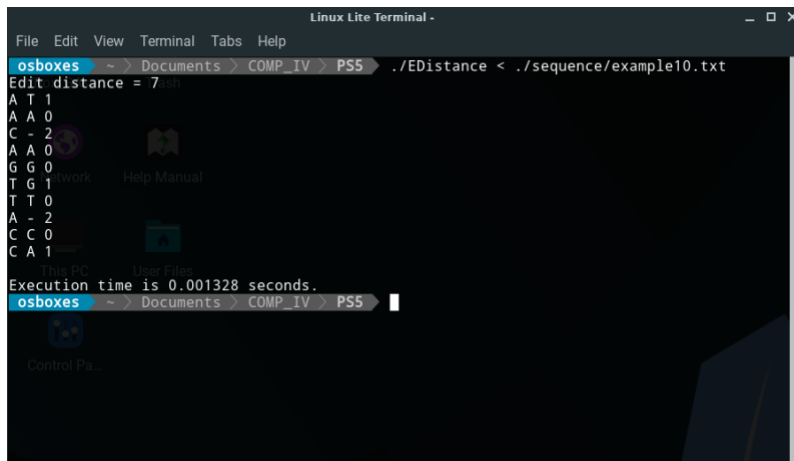
What I Accomplished

My EDistance algorithm uses dynamic programming to fill a matrix with each computation from bottom right to top left, culminating the ideal distance at [0][0]. Tracing the matrix in reverse order from top left to bottom right also shows the alignment path. Instead of computing the same subproblem numerous times, the dynamic programming technique allows the alignment computations to be divided down into subproblems and then stored.

What I Learned

Apart from the numerous applications of Edit Distance, I learned about the advantages of dynamic programming, which, when compared to the recursive solution for this assignment, would have had a much higher space complexity due to the number of recursive calls exceeding $2N$ when comparing two strings of length N . In addition to, I also gained the knowledge on how to use the Valgrind massif tool to efficiently monitor program memory use. This utility shows how much memory the heap consumed during execution. I was able to determine the projected run time and memory use of a bigger sample of strings of length N using the doubling approach.

Output



```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
osboxes ~ - Documents > COMP_IV > PSS ./EDistance < ./sequence/example10.txt
Edit distance = 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
Execution time is 0.001328 seconds.
osboxes ~ - Documents > COMP_IV > PSS
```



```
1: all:
2:     make EDistance
3:
4: EDistance: main.o EDistance.o
5:     g++ -o EDistance -g main.o EDistance.o -lsfml-system
6:
7: main.o: EDistance.cpp main.cpp
8:     g++ -c -g main.cpp -ansi -pedantic -Wall -Werror
9:
10: EDistance.o: EDistance.hpp EDistance.cpp
11:     g++ -c -g -O2 EDistance.cpp -ansi -pedantic
12:
13: clean:
14:     rm -f *.o *~ EDistance
```

```
1: #include <iostream>
2: #include <cstring>
3: #include <SFML/System.hpp>
4:
5: #include "EDistance.hpp"
6:
7: using namespace std;
8:
9: int main() {
10:
11:     // Clock
12:     sf::Clock clock;
13:     sf::Time t;
14:
15:
16:     string String1;
17:     string String2;
18:
19:     // Get input
20:     cin >> String1;
21:     cin >> String2;
22:
23:     // Initialize my class, all math done in constructor and sets relevant
24:     // member variables
25:     EDistance output(String1, String2);
26:
27:
28:     // Get desired output
29:     cout << "Edit distance = " << output.getEditDistance() << endl;
30:     cout << output.getEditString() << endl;
31:
32:     t = clock.getElapsedTime();
33:     cout << "Execution time is " << t.asSeconds() << " seconds." << endl;
34:
35:     return 0;
36: }
```

```
1: #ifndef EDistance_HPP
2: #define EDistance_HPP
3:
4: #include <cstring>
5: #include <vector>
6:
7: class EDistance {
8: public:
9:     EDistance(std::string _stringA, std::string _stringB);
10:     void printOpt() const;
11:     int getEditDistance() const;
12:     std::string getEditString() const;
13:
14: private:
15:     // Input Variables
16:     std::string stringA;
17:     std::string stringB;
18:
19:     // Constructed Variables
20:     std::vector< std::vector<int> > opt;
21:     int editDistance;
22:     std::string editString;
23:
24:     // Private Functions
25:     int optDistance();
26:     std::string alignment() const;
27:     int penalty(char a, char b);
28:     int min(int a, int b, int c);
29: };
30:
31: #endif
```

```
1: #include <iostream>
2: #include <cstring>
3: #include <vector>
4: #include <sstream>
5:
6: #include "EDistance.hpp"
7:
8: using namespace std;
9:
10: EDistance::EDistance(string _stringA, string _stringB) : stringA(_stringA
), stringB(_stringB) {
11:
12:     vector<int> temp;
13:
14:     // Populate the matrix with 0's to start
15:     for(unsigned i = 0; i < stringB.length() + 1; i++)
16:         temp.push_back(0);
17:     for(unsigned i = 0; i < stringA.length() + 1; i++)
18:         opt.push_back(temp);
19:
20:     editDistance = optDistance();
21:     editString = alignment();
22: }
23:
24: int EDistance::penalty(char a, char b) {
25:
26:     if(a == b)
27:         return 0;
28:     else
29:         return 1;
30: }
31: int EDistance::min(int a, int b, int c) {
32:
33:     if(a <= b && a <= c)
34:         return a;
35:     else if(b <= c)
36:         return b;
37:     else
38:         return c;
39: }
40:
41: int EDistance::optDistance() {
42:
43:     // Fill in the matrix with the EditDistances
44:     for(int i = opt.size() - 1; i >= 0; i--)
45:         for(int j = opt[i].size() - 1; j >= 0; j--) {
46:             if((i == opt.size() - 1) && (j == opt[i].size() - 1))
47:                 opt[i][j] = 0;
48:             else if(i == opt.size() - 1)
49:                 opt[i][j] = opt[i][j + 1] + 2;
50:             else if(j == opt[i].size() - 1)
51:                 opt[i][j] = opt[i + 1][j] + 2;
52:             else
53:                 opt[i][j] = min(opt[i + 1][j + 1] + penalty(stringA[i], stringB[j]),
54:                                opt[i + 1][j] + 2,
55:                                opt[i][j + 1] + 2);
56:         }
57:
58:     return opt[0][0];
59: }
60: string EDistance::alignment() const {
61:
62:     stringstream ss;
63:
64:     unsigned i = 0, j = 0;
```

```
65:
66: while(i < opt.size() - 1 || j < opt[0].size() - 1) {
67:     if((i < opt.size() - 1)
68:         && (j < opt[0].size() - 1)
69:         && (opt[i+1][j+1] <= opt[i+1][j] + 1)
70:         && (opt[i+1][j+1] <= opt[i][j+1] + 1)) {
71:         ss << stringA[i] << " " << stringB[j] << " " << opt[i][j] - opt[i+1
][j+1] << '\n';
72:         i++;
73:         j++;
74:     }
75:     else if(((i < opt.size() - 1) && (opt[i+1][j] <= opt[i][j+1]))
76:         || (j == opt[0].size() - 1)) {
77:         ss << stringA[i] << " " << "-" << " " << opt[i][j] - opt[i+1][j] <<
'\n';
78:         i++;
79:     }
80:     else {
81:         ss << "-" << " " << stringB[j] << " " << opt[i][j] - opt[i][j+1] <<
'\n';
82:         j++;
83:     }
84: }
85:
86: return ss.str();
87: }
88:
89: void EDistance::printOpt() const {
90:
91:     // Print the Matrix
92:     for(unsigned i = 0; i < opt.size(); i++) {
93:         for(unsigned j = 0; j < opt[i].size(); j++) {
94:             cout.width(4);
95:             cout << opt[i][j];
96:         }
97:         cout << endl;
98:     }
99: }
100:
101: int EDistance::getEditDistance() const {
102:
103:     return editDistance;
104: }
105:
106: string EDistance::getEditString() const {
107:
108:     return editString;
109: }
```

```
1: /*****
2:  *  readme
3:  *  DNA Sequence Alignment
4:  *****/
5:
6: Name: Matthew Lorette Anaya
7:
8: Hours to complete assignment: 5
9:
10: /*****
11:  * Explain which approach you decided to use when implementing
12:  * (either recursive with memoization, recursive without memoization,
13:  * dynamic programming or Hirschberg\222s algorithm). Also describe why
14:  * you chose this approach and what its pros and cons are.
15:  *****/
16:
17: Implementation of this program was done with the use of dynamic programmi
ng
18: and a matrix. I used the algorithm on the Princeton site in order to fill
19: said matrix. In-order to find the alignment I used backtracking top-left
to
20: bottom right, moving from the current matrix index to the next-lowest mat
rix
21: index. There was a certain case where if the diagonal was 1 higher than t
he
22: downwards or rightwards option, diagonal was still the taken rout. In any
23: case, depending on which direction I went, I either added a gap, or both
24: letters, and incremented i and j counters to traverse back to the bottom
25: right of the matrix.
26:
27:
28:
29:
30: /*****
31:  * Does your code work correctly with the endgaps7.txt test file?
32:  *
33:  * This example should require you to insert a gap at the beginning
34:  * of the Y string and the end of the X string.
35:  *****/
36:
37: Kinda confused here on what this question really is. The pdf is using exa
mple10.txt
38: And this is asking for endgaps7.txt. Seems like there is a mix up of pdfs
between different years of this . So I'm going to use the what the HW pdf says
as there really isn't an example to compare to
39: otherwise. Though it also says to put this all into a folder named ps3, w
hich is definitely incorrect.
40:
41: Input:
42:     Ê./EDistance < ./sequence/example10.txt
43:
44: Expected output:
45:
46:     Edit distance = 7
47:     AT1
48:     AA0
49:     C-2
50:     AA0
51:     GG0
52:     TG1
53:     TT0
54:     A-2
55:     CC0
56:     CA1
```

```
57:
58: What happened:
59:
60:         Edit distance = 7
61:         A T 1
62:         A A 0
63:         C - 2
64:         A A 0
65:         G G 0
66:         T G 1
67:         T T 0
68:         A - 2
69:         C C 0
70:         C A 1
71:
72:         Execution time is 0.00094 seconds.
73:
74:
75:
76:
77: /*****
78:  * Look at your computer's specs in the settings.
79:  * How much RAM does your computer have and explain what this means?
80:  *****/
81:
82: My Mac has 16gb of RAM. Random access memory gives applications a place to
o
83: store and access data on a short-term basis. It stores the information you
ur
84: computer is actively using so that it can be accessed quickly.
85:
86:
87: /*****
88:  * For this question assume M=N. Look at your code and determine
89:  * approximately how much memory it uses in bytes, as a function of
90:  * N. Give an answer of the form  $a * N^b$  for some constants a
91:  * and b, where b is an integer. Note chars are 2 bytes long, and
92:  * ints are 4 bytes long.
93:  *
94:  * Provide a brief explanation.
95:  *
96:  * What is the largest N that your program can handle if it is
97:  * limited to 8GB (billion bytes) of memory?
98:  *****/
99:
100: N^2 is the area of the matrix, the number of integer slots that need to be
e
101: filled in. 4 is the size of an integer in bytes.
102:
103: a = 4
104: b = 2
105: largest N = ~44,721
106:
107: Explanation:
108:         4 * 44,721^2 = 7,999,871,364 just shy of 8gb.
109:
110: /*****
111:  * Run valgrind if you can and attach the output file to your submission.
112:  * If you cannot run it, explain why, and list all errors you're seeing.
113:  * If you can run it successfully, does the memory usage nearly match that
t
114:  * found in the question above?
115:  * Explain why or why not.
```

```

116: /*****
117:
118: -----
119:      n          time(i)          total(B)    useful-heap(B)  extra-heap(B)    sta
cks(B)
120: -----
121: 67  6,808,807,498    3,085,122,584    3,084,577,402        545,182
0
122: 68  6,870,407,882    3,146,681,624    3,146,125,562        556,062
0
123: 69 30,570,148,197    3,201,904,240    3,201,338,395        565,845
0
124:
125: It does not, its actually quite different and I'm not entirely sure as to
why. Not
126: Sure if I'm reading valgrind output wrong or my equation is.
127:
128:
129: /*****
130:  * For each data file, fill in the edit distance computed by your
131:  * program and the amount of time it takes to compute it.
132:  *
133:  * If you get segmentation fault when allocating memory for the last
134:  * two test cases (N=20000 and N=28284), note this, and skip filling
135:  * out the last rows of the table.
136:  *****/
137:
138: data file          distance          time (seconds)
139: -----
140: ecoli2500.txt      118              0.125216
141: ecoli5000.txt      160              0.334861
142: ecoli7000.txt      194              0.521017
143: ecoli10000.txt     223              1.4272
144: ecoli20000.txt     3135             74.6052
145: ecoli28284.txt     8394            177.645
146:
147: /*****
*
148:  * Here are sample outputs from a run on a different machine for
149:  * comparison.
150:  *****/
/
151:
152: data file          distance          time (seconds)
153: -----
154: ecoli2500.txt      118              0.171
155: ecoli5000.txt      160              0.529
156: ecoli7000.txt      194              0.990
157: ecoli10000.txt     223              1.972
158: ecoli20000.txt     3135             7.730
159:
160:
161:
162: /*****
163:  * For this question assume M=N (which is true for the sample files
164:  * above). By applying the doubling method to the data points that you
165:  * obtained, estimate the running time of your program in seconds as a
166:  * polynomial function  $a * N^b$  of N, where b is an integer.
167:  * (If your data seems not to work, describe what went wrong and use
168:  * the sample data instead.)
169:  *
170:  * Provide a brief justification/explanation of how you applied the
171:  * doubling method.

```



```
172:  *
173:  *  What is the largest N your program can handle if it is limited to 1
174:  *  day of computation? Assume you have as much main memory as you need.
175:  *****/
176: a =
177: b =
178: largest N =
179:
180: /
181: *****/
182:  *  Did you use the lambda expression in your assignment? If yes, where
183:  *  (describe a method or provide a lines numbers)
184:  *****/
185: No
186:
187:
188:
189: *****/
190:  *  List whatever help (if any) you received from the course TAs,
191:  *  instructor, classmates, or anyone else.
192:  *****/
193:
194: N/a
195:
196: /*****/
197:  *  Describe any serious problems you encountered.
198:  *****/
199:
200:
201:
202: /*****/
203:  *  List any other comments here.
204:  *****/
205:
```

PS6 Random Writer

For this assignment, we had to create a program to construct a Markov chain that can analyze k-grams (a fixed number of characters) and keep track of the probability of each character's occurrence. The program would then produce text using any input text and a Markov chain of given order k.

Key concepts

The RandWriter class utilizes a map with a kgram key and a map of characters and their frequency. The Mersenne Twister random number generator is also used by RandWriter in the kRand() function. RandWriter constructor creates a map that may be shown as a table-like output, containing each kgram and its frequency, as well as probability for each subsequent letter. When generating a new string from of previously produced characters, the function generate() employs the helper function kRand(), which picks a random next character from a kgram string.

What I Accomplished

Using probabilistic analysis on text to determine the next character/s in a sequence of length k words called kgrams is the Markov Model's name of the game. With the input of a string and order k the RandWriter class maps each of the kgrams in the string to it's following character and frequency. With the given kgram it can then generate a new string based on the probability of each of the following characters. Using this function, the TextWriter class is able to analyze words in text file and generate a pseudorandom string of L length.

What I Learned

Prior to beginning this project, I took it upon myself to absorb more knowledge about how Markov chains are commonly used in systems like online search engines, information retrieval, speech recognition, and gene prediction. I was able to observe firsthand how the Markov model can generate decent text using a trajectory through a table of probabilities of k-grams throughout the assignment.

```
1: CC= g++
2: CFLAGS= -g -Wall -std=c++0x -pedantic
3: Boost= -lboost_unit_test_framework
4:
5: all:
6:     make TextWriter TestWriter
7:
8: TextWriter: RandWriter.o TextWriter.o
9:     $(CC) -o TextWriter TextWriter.o RandWriter.o $(CFLAGS)
10:
11: TestWriter: RandWriter.o Test.o
12:     $(CC) -o TestWriter RandWriter.o Test.o $(Boost)
13:
14: Test.o: RandWriter.h Test.cpp
15:     $(CC) -c Test.cpp $(CFLAGS)
16:
17: TextWriter.o: RandWriter.h TextWriter.cpp
18:     $(CC) -c TextWriter.cpp $(CFLAGS)
19:
20: RandWriter.o: RandWriter.h RandWriter.cpp
21:     $(CC) -c RandWriter.cpp $(CFLAGS)
22:
23: clean:
24:     rm -f *.o *~ TextWriter TestWriter
```

```
1: // Copyright 2022 Matthew Lorette Anaya
2:
3: #include "RandWriter.h"
4: #include <fstream>
5:
6: int main(int argc, char *argv[]) {
7:     if (argc != 3) {
8:         std::cerr << "Usage: ./TextWriter k L < input.txt" << std::endl;
9:         exit(-1);
10:    }
11:    int k = std::atoi(argv[1]);
12:    int L = std::atoi(argv[2]);
13:
14:    int count = 0;
15:    int length = 0;
16:    std::string input;
17:    std::string output;
18:
19:    // read input line by line and generate pseudo-random text
20:    while (std::getline(std::cin, input) && count < L) {
21:        if (input.length() > static_cast<unsigned int>(k)) {
22:            try {
23:                RandWriter rw(input, k);
24:                if (static_cast<int>(input.length()) > L) {
25:                    length = L;
26:                } else if (static_cast<int>(input.length()) + count > L)
{
27:                    length = L - count;
28:                } else {
29:                    length = input.length();
30:                }
31:                output = rw.generate(input.substr(0, k), length);
32:                count += output.length();
33:                std::cout << output << std::endl;
34:            }
35:            catch (std::invalid_argument err) {
36:                std::cerr << err.what() << std::endl;
37:                exit(-1);
38:            }
39:            catch (std::runtime_error err) {
40:                std::cerr << err.what() << std::endl;
41:                exit(-1);
42:            }
43:        }
44:    }
45:
46:    return 0;
47: }
```

```
1: // Copyright 2022 Matthew Lorette Anaya
2: #ifndef RANDWRITER_H
3: #define RANDWRITER_H
4:
5: #include <iostream>
6: #include <string>
7: #include <map>
8: #include <exception>
9: #include <utility>
10: #include <random>
11:
12: class RandWriter {
13: private:
14:
15:     int rw_k;
16:     std::string rw_txt;
17:     std::map<std::string, std::map<char, int>> rw_table;
18:
19: public:
20:
21:     RandWriter(std::string text, int k);
22:
23:     int orderK() const;
24:     int freq(std::string kgram) const;
25:     int freq(std::string kgram, char c) const;
26:
27:     std::string getText() const;
28:     std::string generate(std::string kgram, int L) const;
29:
30:     std::map<std::string, std::map<char, int>> get_table() const;
31:
32:     char kRand(std::string kgram) const;
33:
34:     friend std::ostream& operator<<(std::ostream& out, const RandWriter&
35:     rw);
36:
37: };
38: #endif
```

```
1: // Copyright 2022 Matthew Lorette Anaya
2:
3: #include "RandWriter.h"
4:
5: RandWriter::RandWriter(std::string text, int k) {
6:     rw_txt = text;
7:     rw_k = k;
8:
9:     if (rw_txt.length() < static_cast<unsigned int>(rw_k)) {
10:         throw std::invalid_argument("RandWriter(string text, int k): orde
r k"
11:         " must be less than or equal to text length.");
12:     }
13:
14:     // Table setup
15:     unsigned int pos = 0;
16:     for (unsigned int i = 0; i < rw_txt.length(); i++) {
17:         std::string kgram;
18:         std::map<char, int> freq_table;
19:
20:         // kgrams parsing
21:         for (unsigned int j = i; j < i + rw_k; j++) {
22:             if (j >= rw_txt.length()) {
23:                 pos = j - rw_txt.length();
24:             } else {
25:                 pos = j;
26:             }
27:             kgram += rw_txt.at(pos);
28:         }
29:
30:         // Frequency table setup
31:         pos++;
32:         if (pos >= rw_txt.length()) { pos -= rw_txt.length(); }
33:         freq_table.insert(std::make_pair(rw_txt.at(pos), 0));
34:
35:         // Mapping
36:         if (rw_table.count(kgram) == 0) {
37:             rw_table.insert(std::make_pair(kgram, freq_table));
38:         }
39:
40:         rw_table[kgram][rw_txt.at(pos)]++;
41:     }
42: }
43:
44: int RandWriter::orderK() const {
45:     return rw_k;
46: }
47:
48: std::string RandWriter::getText() const {
49:     return rw_txt;
50: }
51:
52: std::map<std::string, std::map<char, int>> RandWriter::get_table() const
{
53:     return rw_table;
54: }
55:
56: int RandWriter::freq(std::string kgram) const {
57:     if (kgram.length() < static_cast<unsigned int>(rw_k)) {
58:         throw std::runtime_error("freq(string kgram): kgram must be of"
59:         " length greater than or equal to order k.");
60:     }
61:     int count = 0;
62:     for (unsigned int i = 0; i < rw_txt.length(); i++) {
63:         unsigned int pos = 0;
```

```
64:         std::string kg;
65:         // parse input text for kgrams
66:         for (unsigned int j = i; j < i + rw_k; j++) {
67:             // get characters for kgrams
68:             if (j >= rw_txt.length()) {
69:                 pos = j - rw_txt.length();
70:             } else {
71:                 pos = j;
72:             }
73:             kg += rw_txt.at(pos);
74:         }
75:         if (kgram == kg) { count++; }
76:     }
77:     return count;
78: }
79:
80: int RandWriter::freq(std::string kgram, char c) const {
81:     if (kgram.length() < static_cast<unsigned int>(rw_k)) {
82:         throw std::runtime_error("freq(string kgram, char c): kgram must
be"
83:             " of length greater than or equal to order k.");
84:     }
85:     return rw_table.at(kgram).at(c);
86: }
87:
88: char RandWriter::kRand(std::string kgram) const {
89:     if (kgram.length() < static_cast<unsigned int>(rw_k)) {
90:         throw std::runtime_error("kRand(string kgram): kgram must be of"
91:             " length greater than or equal to order k.");
92:     }
93:     if (rw_table.count(kgram) == 0) {
94:         throw std::runtime_error("kRand(string kgram): kgram does not"
95:             " exist.");
96:     }
97:     std::string alphabet;
98:     for (auto const &var1 : rw_table) {
99:         if (var1.first == kgram) {
100:             for (auto const &var2 : var1.second) {
101:                 alphabet += var2.first;
102:             }
103:         }
104:     }
105:     std::random_device device;
106:     std::mt19937 mt_rand(device());
107:     std::uniform_int_distribution<int> distribution(0, alphabet.length()
108:         - 1);
109:
110:     return alphabet[distribution(mt_rand)];
111: }
112:
113: std::string RandWriter::generate(std::string kgram, int L) const {
114:     if (kgram.length() < static_cast<unsigned int>(rw_k)) {
115:         throw std::runtime_error("generate(string kgram, int L): kgram mu
st"
116:             " be of length greater than or equal to order k.");
117:     }
118:     std::string generated = kgram;
119:     // generate new characters based on kgrams
120:     for (int i = rw_k; i < L; i++) {
121:         generated += kRand(generated.substr(i - rw_k, rw_k));
122:     }
123:     return generated;
124: }
125:
126: std::ostream& operator<<(std::ostream& out, const RandWriter& rw) {
```

```
127:     out << "Markov Model\tOrder: " << rw.rw_k << std::endl;
128:     out << "kgram:\tfrequency:\tfreqncy of next char:\tprob of next char:"
<<
129:     std::endl;
130:
131:     for (auto const &var1 : rw.rw_table) {
132:         // var1.first = kgram
133:         out << var1.first << "\t";
134:         out << rw.freq(var1.first) << "\t\t";
135:         for (auto const &var2 : var1.second) {
136:             // var2.first = next char
137:             // var2.second = data
138:             out << var2.first << ":" << var2.second << " ";
139:         }
140:         out << "\t\t\t\t";
141:         for (auto const &var2 : var1.second) {
142:             out << var2.first << ":" << var2.second << "/" <<
143:             rw.freq(var1.first) << " ";
144:         }
145:         out << std::endl;
146:     }
147:     return out;
148: }
```



```
1: // Copyright 2022 Matthew Lorette Anaya
2: #include "RandWriter.h"
3:
4: #define BOOST_TEST_DYN_LINK
5: #define BOOST_TEST_MODULE Main
6: #include <boost/test/unit_test.hpp>
7:
8: BOOST_AUTO_TEST_CASE(base_test) {
9:     std::cout << "***** Test Case 1 *****"
<<
10:     std::endl;
11:
12:     int k = 2;
13:     std::string str = "gagggagaggcgagaaa";
14:     RandWriter rw(str, k);
15:
16:     std::cout << "Printing out Markov Table for string:\n" <<
17:     str << std::endl << std::endl;
18:     std::cout << rw << std::endl;
19:
20:     std::cout << "Testing orderK and freq functions" << std::endl;
21:     BOOST_REQUIRE(rw.orderK() == k);
22:     BOOST_REQUIRE(rw.freq("gg") == 3);
23:     BOOST_REQUIRE(rw.freq("ga", 'g') == 4);
24:
25:     std::cout << "Testing kRand function" << std::endl;
26:     char rand = rw.kRand("aa");
27:     BOOST_REQUIRE(rand == 'a' || rand == 'g');
28:
29:     std::cout << "Testing generate function" << std::endl << std::endl;
30:     BOOST_REQUIRE(rw.generate("ga", 10).length() == 10);
31: }
32:
33: BOOST_AUTO_TEST_CASE(exception_test) {
34:     std::cout << "***** Test Case 2 *****"
<<
35:     std::endl;
36:     std::cout << "Testing construction exception: RandWriter('ADF', 4)" <
<
37:     std::endl;
38:
39:     BOOST_REQUIRE_THROW(RandWriter("ADF", 4), std::invalid_argument);
40:
41:     std::cout << "Testing function exceptions" << std::endl;
42:     RandWriter testMM("abc", 3);
43:     BOOST_REQUIRE_THROW(testMM.freq("a"), std::runtime_error);
44:     BOOST_REQUIRE_THROW(testMM.freq("ab", 'b'), std::runtime_error);
45:     BOOST_REQUIRE_THROW(testMM.kRand("g"), std::runtime_error);
46: }
```

PS7 Kronos – Intro to Regular Expression

In this assignment, we were to utilize regular expressions to parse Kronos InTouch time clock records. We were to check particular lines in the log with regular expressions and report whether a boot was successful or not in a separate file.

Key concepts

This assignment's code was included in a single source file, main.cpp. For both the regex functionality and interacting with the date and time, I utilized the Boost libraries. The only regex I used was to capture the date and time that each boot begun or boot finished line reported. The boot messages themselves were constant and could be scanned without using regex.

What I Accomplished

Depending on the message scanned, the program scans a log file line by line and decides if a boot has started or completed. The date and time are stored and reported if it contains a boot start message. When a boot finished notification is scanned, the date and time, as well as the total boot duration from start to finish, are stored again. This data is stored in a report file (.rpt) that contains each successful and unsuccessful boot attempt.

What I Learned

Despite the fact that this was a short task, I learned the value of regular expressions and how they may be used when scanning for or extracting string/character data. I also discovered how to retrieve the total boot time between the boot start and boot end events in the log using the Boost date/time library.

```
1: CC = g++
2: CFLAGS = -std=c++11 -c -g -O1 -Wall -Werror -pedantic
3: BOOST = -lboost_regex -lboost_date_time
4:
5: all: ps7
6:
7: ps7: main.o
8:      $(CC) -o ps7 main.o $(BOOST)
9: main.o: main.cpp
10:      $(CC) -c main.cpp $(CFLAGS)
11:
12: clean:
13:      rm -f *.o ps7 *~
```

```

1: // Copyright 2022 Matthew Lorette Anaya
2: #include <iostream>
3: #include <fstream>
4: #include <string>
5: #include <exception>
6: #include <boost/regex.hpp>
7: #include <boost/date_time.hpp>
8:
9: using boost::regex;
10: using boost::regex_search;
11: using boost::smatch;
12: using boost::posix_time::ptime;
13: using boost::posix_time::time_duration;
14: using boost::posix_time::time_from_string;
15:
16: int main(int argc, char* argv[]) {
17:
18:     int lineNum = 1,
19:         bootStCnt = 0,
20:         bootDoneCnt = 0;
21:
22:     bool bootStarted = false;
23:
24:     const std::string bootStMsg = "(log.c.166) server started";
25:     const std::string bootDoneMsg = "oejs.AbstractConnector:Started "
26:     "SelectChannelConnector@0.0.0.0:9080";
27:     regex e("^\\d{4}[-](0[1-9]|1[012])[-](0[1-9]|12)[0-9]|3[01])\\s\\d{2}
} "
28:     "[:]\\d{2}[:]\\d{2}");
29:     smatch m;
30:
31:     ptime tBST, tBDT;
32:
33:     std::string s;
34:     std::string fileName;
35:     std::ifstream inFile;
36:     std::ofstream outFile;
37:
38:     if (argc != 2) {
39:         std::cerr << "Usage: ./ps7 device1_intouch.log" << std::endl;
40:         return -1;
41:     }
42:
43:     inFile.open(argv[1]);
44:     if (!inFile.is_open()) {
45:         std::cerr << "Could not open file: " << argv[1] << std::endl;
46:         return -1;
47:     }
48:     s = fileName = argv[1];
49:     outFile.open(s.append(".rpt.tmp"));
50:     fileName = fileName.substr(fileName.find_last_of("\\/") + 1);
51:
52:     // Temp report file = scanned boot
53:     while (std::getline(inFile, s)) {
54:         if (bootStarted) {
55:             if (s.find(bootDoneMsg) != std::string::npos) {
56:                 // Boot Done
57:                 regex_search(s, m, e);
58:                 tBDT = ptime(time_from_string(m[0]));
59:                 time_duration td = tBDT - tBST;
60:
61:                 outFile << lineNum << "(" << fileName << ")" << m[0]
62:                 << " Boot Completed" << std::endl
63:                 << "\tBoot Time: " << td.total_milliseconds() << "ms"
64:                 << std::endl << std::endl;

```

```
65:
66:         bootStarted = false;
67:         bootDoneCnt++;
68:     } else if (s.find(bootStMsg) != std::string::npos) {
69:         // Failed boot
70:         regex_search(s, m, e);
71:         tBST = ptime(time_from_string(m[0]));
72:
73:         outFile << "**** Incomplete boot ****" << std::endl <<
74:         std::endl
75:         << "=== Device boot ===" << std::endl
76:         << lineNum << "(" << fileName << ")" " " << m[0]
77:         << " Boot Start" << std::endl;
78:
79:         bootStCnt++;
80:     }
81: } else if (s.find(bootStMsg) != std::string::npos) {
82:     // Successfull boot
83:     regex_search(s, m, e);
84:     tBST = ptime(time_from_string(m[0]));
85:     outFile << "=== Device boot ===" << std::endl
86:     << lineNum << "(" << fileName << ")" " " << m[0]
87:     << " Boot Start" << std::endl;
88:     bootStarted = true;
89:     bootStCnt++;
90: }
91: lineNum++;
92: }
93: inFile.close();
94: outFile.close();
95:
96: // Report file done
97: s = argv[1];
98: s.append(".rpt");
99: outFile.open(s);
100:
101: s.append(".tmp");
102: inFile.open(s);
103: if (!inFile.is_open()) {
104:     std::cerr << "Could not open file: " << s << std::endl;
105:     return -1;
106: }
107:
108: outFile << "Device Boot Report" << std::endl << std::endl
109: << "InTouch log file: " << fileName << std::endl
110: << "Lines Scanned: " << lineNum - 1 << std::endl << std::endl
111: << "Device boot count: initiated: " << bootStCnt << ", completed: "
112: << bootDoneCnt << std::endl << std::endl << std::endl;
113:
114: outFile << inFile.rdbuf();
115: inFile.close();
116: outFile.close();
117:
118: // remove temp report file
119: if (std::remove(s.c_str()) != 0) {
120:     std::cerr << "Error deleting temp file: " << s << std::endl;
121:     return -1;
122: }
123:
124: return 0;
125: }
```