

# COMP2400 Project Portfolio

Emily Sheehan

May-June 2022

**PS0: Hello World with SFML**

**PS1a: Linear Feedback Shift Register**

**PS1b: PhotoMagic**

**PS2: Triangle Fractal**

**PS3a: N-Body Simulation (Static)**

**PS3b: N-Body Simulation**

**PS4a: CircularBuffer**

**PS4b: String Sound**

**PS5: DNA String Alignment**

**PS6: Random Writer**

**PS7: Kronos Log Parsing**

## Portfolio Statement

The following document contains a synopsis of my entire semester's worth of work in COMP2400 at UMass Lowell. Over the course of the semester, I completed 10 new projects, wrote several thousand lines of code, worked with at least 21 new libraries, and learned more in 6 weeks than ever before. I hope that this portfolio illustrates my growth and learning throughout our short time together. I sincerely enjoyed the class and appreciate all of your patience, guidance, and debugging!

## PS0: Hello World with SFML

### PROJECT SUMMARY AND WHAT I LEARNED

This project encompassed both the build environment set-up as well as an introduction to C++ graphics using SFML. Firstly, we had to get the Linux environment up and running. For my machine, I ended up choosing the WSL (Windows Subsystem for Linux) route after struggling to get VirtualBox, a VM manager, working. Once I had figured out my Linux environment, it was time to start programming. We followed the simple green circle tutorial provided by <https://www.sfml-dev.org/tutorials/2.4/start-linux.php> and then extended it by implementing a sprite that moved around the screen with key-presses (arrow keys). I also added an additional feature that allowed for the resizing of the sprite with keystrokes as well (the comma and period keys). Additionally, I changed the green circle (from the initial tutorial) to a pink hexagon.

### KEY ALGORITHMS, DATA STRUCTURES AND/OR OO DESIGNS

This program was all contained within one function, `main()`, that handled both the graphical display as well as the key presses. After creating and sizing the window and creating the Sprite object, I created booleans for each key to indicate whether they had been pressed or not. I then had two nested while loops that handled the events: rendering the graphics while the window was open and then processing the pressing and releasing of a key. I used the `sf::Keyboard` library to handle key presses, and used a series of cases in order to distinguish between which key was pressed. Upon each press, I toggled the corresponding boolean to true/false. I then had a series of if statements that, if an arrow key was pressed, would alter the x/y coordinate by  $\pm 2$ . For the additional feature, I chose to incorporate a size scaler for the sprite because I had to scale the sprite's size to begin with in order to fit in the screen, so it was an easy addition. I incorporated this via key presses as well, and instead of changing the coordinate of the sprite upon a key press, I instead just called the `scale()` function and increased/decreased the sprite's size by  $\pm 1$ .

### SCREENSHOT OF COMPILED PROGRAM



## main.cpp

---

```
1 //Emily Sheehan
2 //To move the sprite, use the arrow keys. To increase the sprites size, press the '.' key (period). To
   decrease its size, press the ',' (comma).
3
4 #include <SFML/Graphics.hpp>
5
6 int main()
7 {
8
9     bool upArrowPressed=false;
10    bool downArrowPressed=false;
11    bool leftArrowPressed=false;
12    bool rightArrowPressed=false;
13    bool periodPressed=false;
14    bool commaPressed=false;
15
16    sf::RenderWindow window(sf::VideoMode(500, 500), "Emily's Project 0");
17    sf::CircleShape shape(80, 8);
18    shape.setFillColor(sf::Color::Magenta);
19
20    //values to set sprite postion
21    int x_coord = window.getSize().x / 2;
22    int y_coord = window.getSize().y / 2;
23
24    sf::Texture texture;
25    if (!texture.loadFromFile("sprite.png"))
26        return EXIT_FAILURE;
27
28    sf::Sprite sprite(texture);
29    sprite.setScale(.05,.05);
30
31    while (window.isOpen())
32    {
33        sf::Event event;
34
35        while (window.pollEvent(event))
36        {
37            if (event.type == sf::Event::Closed)
38                window.close();
39            if (event.type == sf::Event::KeyPressed)
40            {
41                switch (event.key.code)
42                {
43                    // If escape is pressed, close the application
44                    case sf::Keyboard::Escape : window.close(); break;
45
46                    //Handles the up, down, left,right, period, and comma key presses
47                    case sf::Keyboard::Up:
48                        upArrowPressed=true;
49                        break;
50                    case sf::Keyboard::Down:
51                        downArrowPressed=true;
52                        break;
53                    case sf::Keyboard::Left:
54                        leftArrowPressed=true;
55                        break;
56                    case sf::Keyboard::Right:
57                        rightArrowPressed=true;
58                        break;
59                    case sf::Keyboard::Period:
60                        periodPressed=true;
61                        break;
62                    case sf::Keyboard::Comma:
```

```

63         commaPressed=true;
64         break;
65     default : break;
66     }
67 }
68
69 // If a key is released
70 if (event.type == sf::Event::KeyReleased)
71 {
72     switch (event.key.code)
73     {
74         //Handles the up, down, left,right, period, and comma key releases
75
76         case sf::Keyboard::Up :
77             upArrowPressed=false;
78             break;
79         case sf::Keyboard::Down:
80             downArrowPressed=false;
81             break;
82         case sf::Keyboard::Left:
83             leftArrowPressed=false;
84             break;
85         case sf::Keyboard::Right:
86             rightArrowPressed=false;
87             break;
88         case sf::Keyboard::Period:
89             periodPressed=false;
90             break;
91         case sf::Keyboard::Comma:
92             commaPressed=false;
93             break;
94
95         default : break;
96     }
97 }
98 }
99 //if key pressed...
100
101 if (leftArrowPressed){
102     x_coord -= 2;
103 }
104 else if (rightArrowPressed){
105     x_coord += 2;
106 }
107 else if (upArrowPressed){
108     y_coord -= 2;
109 }
110 else if (downArrowPressed){
111     y_coord += 2;
112 }
113 else if(periodPressed){
114     sprite.scale(1.01, 1.01);
115 }
116 else if(commaPressed){
117     sprite.scale(.99, .99);
118 }
119 window.clear();
120
121 sprite.setPosition(x_coord, y_coord);
122
123 window.draw(shape);
124 window.draw(sprite);
125 window.display();https://www.overleaf.com/project/62b9a13f9032d28334dcb0e8
126 }
127

```

```
128     return 0;
129 }
```

---

## Makefile

---

```
1 CC = g++
2 CFLAGS = --std=c++14 -Wall -Werror -pedantic
3 LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
4
5 all: sfml-app
6
7 %.o: %.cpp $(DEPS)
8     $(CC) $(CFLAGS) -c $<
9
10 sfml-app: main.o
11     $(CC) $(CFLAGS) -o $@ $^ $(LIB)
12
13 clean:
14     rm *.o sfml-app
```

---

## PS1a: Linear Feedback Shift Register

### PROJECT SUMMARY AND WHAT I LEARNED

This project produces "random" bits by implementing a Fibonacci linear feedback shift register. A LFSR works by shifting the bits of the input to the left by one and then replacing the empty bit by the bit at tap position. The project is structured around a constructor, two functions, and an insertion operator. The constructor takes an initial String, seed, whose characters are a series of 0 and 1. The first function, generate(), takes an int k as a parameter and simulates k steps through the register and then returns a k-bit integer. Step() also simulates one step and returns a bit (0 or 1). My constructor operator verifies that the input was valid (no non-zero or non-one digits, 16 bits), and then pushed back each digit onto a vector that was then utilized in step() and generate(). In addition to the LFSR itself, we also learned about unit testing with the Boost framework for this project. In my tests, I utilized BOOST\_REQUIRE\_EQUAL to ensure that my step() function was working correctly by comparing the value returned on its call with the expected value. I followed a similar process for generate() by comparing its return value with the expected. I also tested against invalid sequences using BOOST\_REQUIRE and ensuring that an error message was thrown.

### KEY ALGORITHMS, DATA STRUCTURES AND/OR OO DESIGNS

The core of the project I created my FibLFSR using a bool vector representing each bit. For my step function, I got the left most bit (@ pos 0 in the vector) and x-ored it with the tap bits (@ pos 2,3,5). I then shifted each "bit" except for the leftmost in the vector (true or false) by one to the left, set the first bit equal to the left most bit, and returned the new leftbit. For my generate function, I set an int equal to 0 to begin with and then iterated k times, each time multiplying the int by 2 and adding it to the int that step() returned. In my ;; operator, I first created two checks to ensure that the initial input was a 16 bit sequence containing only 1s and 0s. If so, I then printed the output.

### SCREENSHOT OF COMPILED PROGRAM

```
g++ test.o FibLFSR.o -o test -lboost_unit_test_framework
root@EmilyXPS:~/COMP2400/ps1a# ./test
Running 4 test cases...

*** No errors detected
root@EmilyXPS:~/COMP2400/ps1a#
```



## FibLFSR.hpp

---

```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4
5 using namespace std;
6
7 class FibLFSR {
8     public:
9         // Constructor to create LFSR with the given initial seed
10        FibLFSR(string seed);
11        // Simulate one step and return the new bit as 0 or 1
12        int step();
13        // Simulate k steps and return a k-bit integer
14        int generate(int k);
15        vector<bool> tap_vector;
16        unsigned int seed_size;
17        bool wrong_digit;
18        ~FibLFSR();
19    private:
20
21    // Any fields that you need
22    };
23 std::ostream& operator<<(std::ostream&, const FibLFSR& flfsr);
```

---

## FibLFSR.cpp

---

```
1 #include "FibLFSR.hpp"
2 #include <iostream>
3 #include <string>
4 #include <vector>
5
6 using namespace std;
7
8 FibLFSR::FibLFSR(string seed)
9 {
10     //num_taps = 3;
11     seed_size = seed.length();
12
13
14     for(char& c : seed){
15         if(c=='1'){
16             tap_vector.push_back(true);
17         }
18         else if (c=='0'){
19             tap_vector.push_back(false);
20         }
21         else{
22             wrong_digit = true;
23         }
24     }
25 }
26
27
28 FibLFSR::~FibLFSR() {}
29
30 // shifts the seed one to the left by one bit
31 int FibLFSR::step()
32 {
33     if(tap_vector.size() != 16){
34         return -1;
35     }
36     else if(wrong_digit == true){
```

```

37     return -1;
38 }
39
40 int bit1;
41 int bit2;
42 int bit3;
43
44 int leftbit = tap_vector[0];
45
46 bit1 = tap_vector[2];
47 bit2 = tap_vector[3];
48 bit3 = tap_vector[5];
49
50 leftbit = ((bit1 ^ leftbit) ^ bit2) ^ bit3;
51
52 //shifts bits except for last
53 for (unsigned i = 0; i < (15); i++){
54     tap_vector[i] = tap_vector[i+1];
55 }
56
57 tap_vector[15] = leftbit;
58
59 // cout << *this << endl;
60 return leftbit;
61 }
62 }
63
64 int FibLFSR::generate(int k)
65 {
66     int x = 0;
67     for (int i = 0; i < k; i++){
68         x = x * 2;
69         x = x + step();
70     }
71     return x;
72 }
73 }
74
75 ostream &operator<<(ostream &output, FibLFSR const &flfsr)
76 {
77     if(flfsr.wrong_digit == true){
78         output << "crazy input - contains a non-0 or 1 value";
79         return output;
80     }
81     else if(flfsr.seed_size != 16){
82         output << "Not 16 Bits";
83         return output;
84     }
85
86     for (unsigned i = 0; i < flfsr.tap_vector.size(); i++)
87         if (flfsr.tap_vector[i] == true)
88         {
89             output << 1;
90         }
91
92     else if (flfsr.tap_vector[i] ==false)
93     {
94         output << 0;
95     }
96     return output;
97 }
98 }
99

```

---

## Makefile

---

```
1 CC= g++
2 CFLAGS= --std=c++17 -Wall -Werror -pedantic
3 Boost= -lboost_unit_test_framework
4 DEPS = FibLFSR.hpp
5
6 # Make both projects
7 all: ps1 test
8
9 %.o: %.cpp $(DEPS)
10     $(CC) $(CFLAGS) -c $<
11
12 ps1: test.o FibLFSR.o
13     $(CC) test.o FibLFSR.o -o ps1 $(Boost)
14
15 test: test.o FibLFSR.o
16     $(CC) test.o FibLFSR.o -o test $(Boost)
17
18 clean:
19     rm *.o
20     rm ps1
21     rm test
```

---

## PS1b: PhotoMagic

### PROJECT SUMMARY AND WHAT I LEARNED

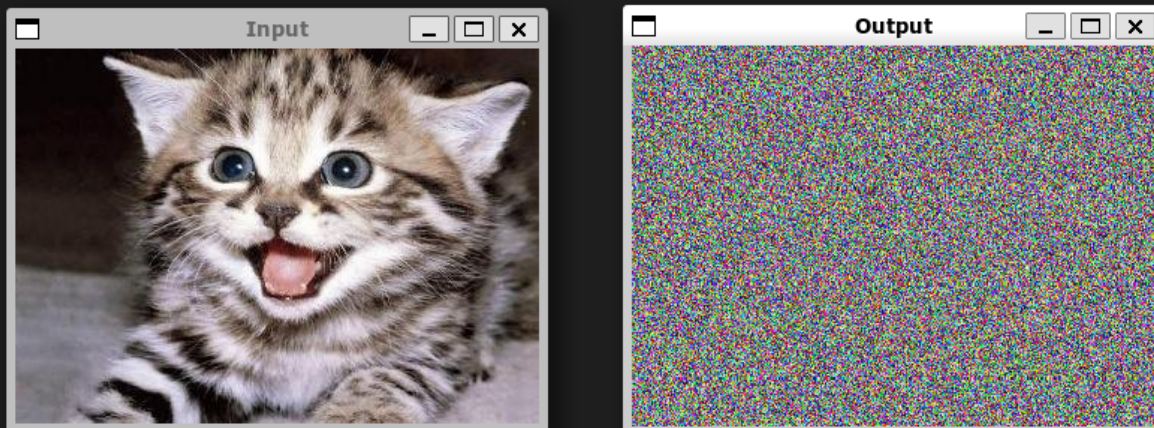
This project implements the Fibonacci LFSR created in project 1a and uses it to encrypt an image provided by the user and displays this image using the SFML library. As an extension, it also implements a two digit password instead of the traditional 16 digit seed. The program accepts four arguments: the name of the program (PhotoMagic), the input file, the output, and the initial 2 character "password". It converts this password to binary, creates a FibLFSR object, and then cycles through it xors the rgb values of each pixel in the image with the number from calling generate. It then displays the transformed photo on the screen using the same sfml library we used in project 0.

### KEY ALGORITHMS, DATA STRUCTURES AND/OR OO DESIGNS

The photo-altering occurs via a transform method. The function accepts a reference to an image along with a FibLFSR object. It then gets the vector created by the FibLFSR object, and it iterates through each pixel in the image using the dimensions. It grabs each pixel and gets its rgb values. Each value is then XOR-ed with the value returned by calling the generate() function on the k-value 15. With these three new numbers, the pixel in the image is set to this value. This function is called from within the main(). Main(), per usual, handles the display and overall organization of the program. It takes the 3 provided command line arguments (input file, output file, and password). It then converts the letter password to a 16 digit password using each letter's 8 bit binary code. After this, main() calls transform, renders the two windows, and displays them to the screen.

### SCREENSHOT OF COMPILED PROGRAM

```
root@EmilyXPS:~/COMP2400/ps1b# ./PhotoMagic cat.jpg encrypted.png ab
0110000101100010
Setting vertical sync not supported
```



FibLFSR.hpp, FibLFSR.cpp (see ps1a)  
photomagic.cpp

---

```
1  #include "FibLFSR.hpp"
2
3  #include <SFML/System.hpp>
4  #include <SFML/Window.hpp>
5  #include <SFML/Graphics.hpp>
6  #include <bitset>
7
8  void transform(sf::Image &img, FibLFSR *fib);
9
10 using namespace std;
11
12 int main(int argc, char *argv[])
13 {
14     if(argc != 4){
15         cout << "incorrect number of arguments-- should be 4" << endl;
16         return -1;
17     }
18
19     string input_file = argv[1];
20     string output_file = argv[2];
21     string password = argv[3];
22     string FibLFSR_seed;
23
24     for(char& c : password){
25         FibLFSR_seed += bitset<8>(c).to_string();
26     }
27
28     cout<<FibLFSR_seed<< endl;
29
30
31     FibLFSR fib(FibLFSR_seed);
32     sf::Image img_input;
33
34     if (!img_input.loadFromFile(input_file))
35     {
36         return -2;
37     }
38
39     sf::Image img_output = img_input;
40
41
42     transform(img_output, &fib);
43
44     sf::RenderWindow window_input(sf::VideoMode(img_input.getSize().x, img_input.getSize().y), "Input");
45     sf::RenderWindow window_output(sf::VideoMode(img_output.getSize().x, img_output.getSize().y), "Output");
46
47     sf::Texture texture_input;
48     texture_input.loadFromImage(img_input);
49     sf::Texture texture_output;
50     texture_output.loadFromImage(img_output);
51
52     sf::Sprite sprite_input;
53     sprite_input.setTexture(texture_input);
54     sf::Sprite sprite_output;
55     sprite_output.setTexture(texture_output);
56
57     while (window_input.isOpen() && window_output.isOpen())
58     {
59         sf::Event event;
60         while (window_input.pollEvent(event))
61         {
```

```

62         if (event.type == sf::Event::Closed)
63             window_input.close();
64     }
65     while (window_output.pollEvent(event))
66     {
67         if (event.type == sf::Event::Closed)
68             window_output.close();
69     }
70     window_input.clear();
71     window_input.draw(sprite_input);
72     window_input.display();
73     window_output.clear();
74     window_output.draw(sprite_output);
75     window_output.display();
76 }
77
78 if(!img_output.saveToFile(output_file)){
79     cout << "could not save to file" << endl;
80     return -3;
81 }
82
83
84 return 0;
85
86 }
87
88 void transform(sf::Image &img, FibLFSR *fib)
89 {
90     sf::Color p;
91     sf::Vector2u size = img.getSize();
92     vector<bool> fib_vector = fib->get_fib_vector();
93
94     for (unsigned int x = 0; (x < size.x); x++)
95     {
96         for (unsigned int y = 0; (y < size.y); y++)
97         {
98             p = img.getPixel(x, y);
99
100             p.r = p.r ^ fib->generate(15);
101             p.g = p.g ^ fib->generate(15);
102             p.b = p.b ^ fib->generate(15);
103
104             img.setPixel(x, y, p);
105         }
106     }
107 }

```

---

## Makefile

---

```
1 CC= g++
2 CFLAGS= --std=c++17 -Wall -Werror -pedantic
3 DEPS = FibLFSR.hpp
4 LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
5
6
7 # Make both projects
8 all: PhotoMagic test
9
10 %.o: %.cpp $(DEPS)
11     $(CC) $(CFLAGS) -c $<
12
13 PhotoMagic: photomagic.o FibLFSR.o
14     $(CC) photomagic.o FibLFSR.o -o PhotoMagic $(LIB)
15
16 test: test.o FibLFSR.o
17     $(CC) test.o FibLFSR.o -o test $(LIB)
18
19 clean:
20     rm *.o
21     rm PhotoMagic
22     rm test
```

---

## PS2: TFractal

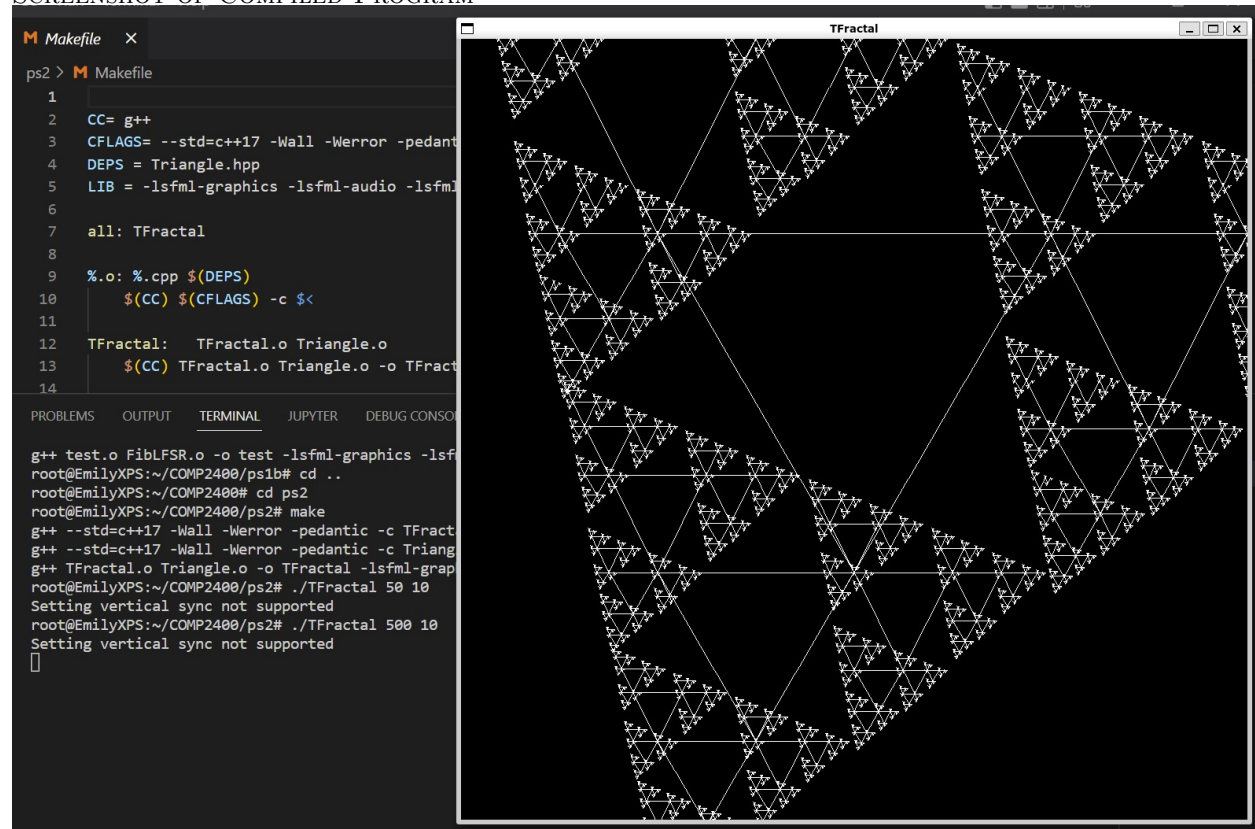
### PROJECT SUMMARY AND WHAT I LEARNED

Provided a length (base of equilateral triangle) and depth ( of recursions), this project draws fractal triangles. It also has a function to change the background color of the drawing upon pressing the space bar. The program is built off of a triangle class that implements the logic for drawing a triangle as well as its descendants. There is then a file called TFractal which contains a recursive function fTree() and main() which draws and displays the triangles. We also began to lint for this assignment, using google's style guide for C++. Quite frankly, this project is a little messy and imperfect- I was on vacation so was rushed and did not complete it to the best of my ability. At the time, I chose to draw the triangle using lines rather than using a shape derivative because the lines seemed less complicated. As a result, the triangles are not equilateral and the window does not size to the image. The triangles I created using lines rather than via the polygon shape in the SFML library, so it was a careless error. I also just forgot to size the image properly. Despite this, I did manage to regain an extra credit point by changing the window's background color.

### KEY ALGORITHMS, DATA STRUCTURES AND/OR OO DESIGNS

As mentioned above, the project is based off of a triangle class which handles the graphics for drawing a triangle with the proper angles. A triangle object is then passed into fTree() along with an int depth, int length, and a RenderWindow window. If the depth is  $\neq 0$  at the top of the call, the function returns. Otherwise, it creates three triangle objects using the triangle provided, draws them to the screen, subtracts 1 from the depth, and then calls itself on each of the three triangles it created. Within the main(), it creates a window, an initial triangle (dimensions to match the middle of the screen), and calls fTree() on it. It then handles the background color changing using key-presses. If a space is detected, the window clears and is set to a random color.

### SCREENSHOT OF COMPILED PROGRAM





## Triangle.hpp

---

```
1 // Copyright 2022 Emily Sheehan
2
3 #include <SFML/System.hpp>
4 #include <SFML/Window.hpp>
5 #include <SFML/Graphics.hpp>
6
7 #ifndef TRIANGLE_HPP_
8 #define TRIANGLE_HPP_
9
10 class Triangle: public sf::Drawable {
11 public:
12     Triangle(sf::Vector2f, sf::Vector2f, sf::Vector2f);
13
14     sf::Vector2f angle1;
15     sf::Vector2f angle2;
16     sf::Vector2f angle3;
17
18     Triangle *tri1;
19     Triangle *tri2;
20     Triangle *tri3;
21
22     double length;
23 private:
24     void draw(sf::RenderTarget &target, sf::RenderStates states) const;
25 };
26 #endif // TRIANGLE_HPP_
```

---

## Triangle.cpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #include "/root/COMP2400/ps2/Triangle.hpp"
4
5 Triangle::Triangle(sf::Vector2f one, sf::Vector2f two, sf::Vector2f three) {
6     angle1 = one;
7     angle2 = two;
8     angle3 = three;
9 }
10
11 void Triangle::draw(sf::RenderTarget &rt, sf::RenderStates s) const {
12     sf::Vertex line1[] = {sf::Vertex(angle1), sf::Vertex(angle2)};
13     sf::Vertex line2[] = {sf::Vertex(angle2), sf::Vertex(angle3)};
14     sf::Vertex line3[] = {sf::Vertex(angle1), sf::Vertex(angle3)};
15
16     rt.draw(line1, 2, sf::Lines);
17     rt.draw(line2, 2, sf::Lines);
18     rt.draw(line3, 2, sf::Lines);
19 }
```

---

## TFractal.cpp

---

```

1  // Copyright 2022 Emily Sheehan
2  #include <iostream>
3  #include <vector>
4  #include <cmath>
5
6  #include "/root/COMP2400/ps2/Triangle.hpp"
7
8
9
10 void fTree(Triangle t, int depth, int length, sf::RenderWindow *window) {
11     if (depth <= 0) {
12         return;
13     }
14
15     sf::Vector2f t1a(t.angle1.x - length/2, t.angle1.y - length);
16     sf::Vector2f t1b(t.angle1.x + length/2, t.angle1.y - length);
17     t.tri1 = new Triangle(t1a, t1b, t.angle1);
18
19     sf::Vector2f t2a(t.angle2.x + length, t.angle2.y);
20     sf::Vector2f t2b(t.angle2.x + length/2, t.angle2.y + length);
21     t.tri2 = new Triangle(t.angle2, t2a, t2b);
22
23     sf::Vector2f t3a(t.angle3.x - length, t.angle3.y);
24     sf::Vector2f t3b(t.angle3.x - length / 2,
25                     t.angle3.y + (length * sqrt(3) / 2));
26     t.tri3 = new Triangle(t3a, t.angle3, t3b);
27
28     window->draw(*t.tri3);
29     window->draw(*t.tri1);
30     window->draw(*t.tri2);
31
32     depth -= 1;
33
34     fTree(*t.tri1, depth, length/2, window);
35     fTree(*t.tri2, depth, length/2, window);
36     fTree(*t.tri3, depth, length/2, window);
37 }
38
39 int main(int argc, char *argv[]) {
40     sf::RenderWindow window(sf::VideoMode(1000, 1000), "TFractal");
41
42     double length = std::stod(argv[1]);
43     int depth = atoi(argv[2]);
44
45     sf::Vector2f tri_a(500 - length / 2, 500 - length / 2);
46     sf::Vector2f tri_b(500 - length / 2 + length, 500 - length / 2);
47     sf::Vector2f tri_c(500 - length / 2 + length / 2,
48                       500 - length / 2 + (length * sqrt(3) / 2));
49     Triangle tr(tri_a, tri_b, tri_c);
50
51     window.draw(tr);
52     fTree(tr, depth, length / 2, &window);
53
54     sf::Texture tex;
55     tex.create(1000, 1000);
56     tex.update(window);
57
58     sf::Texture tex2;
59     tex2.loadFromImage(tex.copyToImage());
60
61     sf::Sprite s;
62     s.setTexture(tex2);
63

```

```

64     while (window.isOpen()) {
65         sf::Event event;
66
67         while (window.pollEvent(event)) {
68             if (event.type == sf::Event::Closed)
69                 window.close();
70
71             if (event.type == sf::Event::KeyPressed) {
72                 switch (event.key.code) {
73                     // If escape is pressed, close the application
74                     case sf::Keyboard::Escape:
75                         window.close();
76                         break;
77
78                     // change color on space press
79                     case sf::Keyboard::Space:
80                         window.clear(sf::Color(rand()%256, rand()%256, rand()%256));
81                         break;
82
83                     default:
84                         break;
85                 }
86             }
87             window.draw(s);
88             window.display();
89         }
90     }
91 }

```

---

## Makefile

---

```
1
2 CC= g++
3 CFLAGS= --std=c++17 -Wall -Werror -pedantic
4 DEPS = Triangle.hpp
5 LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
6
7 all: TFractal
8
9 %.o: %.cpp $(DEPS)
10     $(CC) $(CFLAGS) -c $<
11
12 TFractal: TFractal.o Triangle.o
13     $(CC) TFractal.o Triangle.o -o TFractal $(LIB)
14
15 lint:
16     cpplint --filter=--runtime/references,-build/c++11 --root=. *.cpp *.hpp
17
18 clean:
19     rm *.o
20     rm TFractal
21     rm *.gch
```

---

## PS3a: NBody Simulation (static)

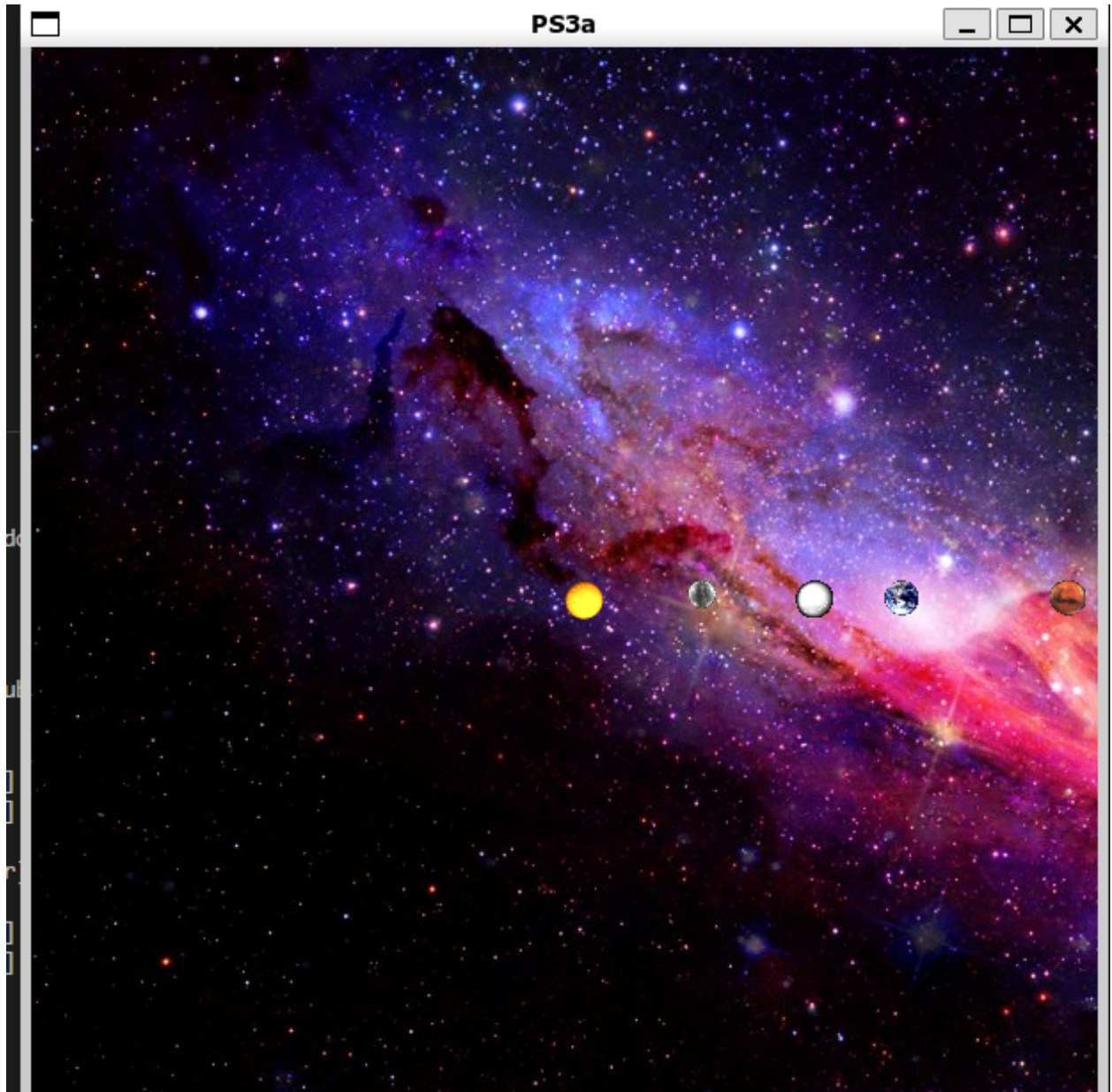
### PROJECT SUMMARY AND WHAT I LEARNED

In my opinion, this was the most complicated project because it required a lot of moving parts. This project models a "universe" and the celestial bodies within that universe to scale. It serves as the first step in a two part project intended to solve Sir Isaac Newton's question of n-particle motion. The project works by reading in celestial body objects whose data is provided via a txt file specified by the user in the command line. There is a `CelestialBody` class too and a universe object contains all such instances. The celestial body class creates such objects by performing dimensional analysis and converting the x position of the celestial body from the scale of the universe to the scale of the window. The universe then handles the portrayal of the bodies. Once all celestial objects are displayed, the background galaxy image is loaded in too. In this project, we also implemented the concept of being memory-safe, and I used pointers to celestial bodies inside of my universe object rather than celestial bodies themselves.

### KEY ALGORITHMS, DATA STRUCTURES AND/OR OO DESIGNS

As mentioned above, the project is based off of two classes, `CelestialBody` and `Universe`. `CelestialBody` contains the framework for each object in the universe- its dimensions, positions, velocity, and mass. It also has an image, sprite, and texture within its constructor. It contains two operators, to read in from the text file and print out to the screen. This class is fairly straight forward as it simply creates an object based off of the provided dimensions and renders its image. Then, a `Universe` object is created to hold each `CelestialBody`. It accepts an int, num planets, and a float, radius. It also has a vector of `CelestialBody` pointers, a virtual draw function, and overrides the input and output operators. In `main()`, the text file is read in, the celestial bodies are read in from the text file, an instance of the `Universe` is created, and the draw function is then called from within to display the image. This project once again uses the SFML library for graphics. When I sat down to write this section of my project, I felt dread because I remember this project being quite difficult. In retrospect, it was not too difficult but again it had a lot of components that had to work together. At the time, I worked from the big picture down to the element, instead of the other way around. That is one of my biggest take-aways of the semester- make the little components and then expand to the bigger ones.

### SCREENSHOT OF COMPILED PROGRAM



## CelestialBody.hpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #pragma once
4
5 #include <iostream>
6 #include <string>
7 #include <fstream>
8 #include <vector>
9 #include <SFML/Graphics.hpp>
10 #include <SFML/Window.hpp>
11
12 class CelestialBody: public sf::Drawable {
13 private:
14     void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;
15     double universe_radius, window_radius, xpos, ypos, xvel, yvel, mass;
16     std::string img_name;
17     sf::Image image;
18     sf::Sprite sprite;
19     sf::Texture texture;
20
21 public:
22     friend std::istream& operator>> (std::istream &input, CelestialBody &CB);
23     friend std::ostream& operator<< (std::ostream &output, CelestialBody &CB);
24     explicit CelestialBody(double universe_radius, double window_radius = 300);
25 };
26
27 std::istream& operator>> (std::istream &input, CelestialBody &CB);
28 std::ostream& operator<< (std::ostream &output, CelestialBody &CB);
```

---

## CelestialBody.cpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #include "CelestialBody.hpp"
4
5 void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states)
6 const {
7     target.draw(sprite);
8 }
9
10 CelestialBody::CelestialBody(double universe_radius, double window_radius)
11 : universe_radius(universe_radius), window_radius(window_radius) {
12 }
13
14 std::istream& operator>> (std::istream &input, CelestialBody &CB) {
15     input >> CB.xpos >> CB.ypos;
16     input >> CB.xvel >> CB.yvel;
17     input >> CB.mass >> CB.img_name;
18
19     if (!CB.image.loadFromFile(CB.img_name)) {
20         throw std::ios_base::failure("Couldn't load image");
21     }
22
23     CB.texture.loadFromImage(CB.image);
24
25     CB.sprite.setTexture(CB.texture);
26
27     double x = (1 + CB.xpos/CB.universe_radius) * CB.window_radius;
28     double y = ((1-CB.ypos/CB.universe_radius) * CB.window_radius);
29     CB.sprite.setPosition(sf::Vector2f(x, y));
30
31     return input;
```

```
32 }
33
34 std::ostream& operator<< (std::ostream &output, CelestialBody &CB) {
35     output << "Name: " << CB.img_name << std::endl;
36     output << "X Position: " << CB.xpos << std::endl;
37     output << "Y Position: " << CB.ypos << std::endl;
38     output << "X Velocity: " << CB.xvel << std::endl;
39     output << "Y Velocity: " << CB.yvel << std::endl;
40     output << "Mass: " << CB.mass << std::endl;
41
42     return output;
43 }
```

---



## Universe.hpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #pragma once
4
5 #include <iostream>
6 #include <string>
7 #include <fstream>
8 #include <vector>
9 #include <SFML/Graphics.hpp>
10 #include <SFML/Window.hpp>
11 #include "CelestialBody.hpp"
12
13 class Universe : public sf::Drawable {
14 private:
15     unsigned int n;
16     float r;
17     std::vector<CelestialBody*> c_bodies;
18     void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;
19
20 public:
21     explicit Universe(unsigned int num_planets = 0, float radius = 0);
22
23     friend std::istream& operator>> (std::istream &input, Universe &u);
24     friend std::ostream& operator<< (std::ostream &output, Universe &u);
25
26     ~Universe();
27 };
28
29 std::istream& operator>> (std::istream &input, Universe &u);
30 std::ostream& operator<< (std::ostream &output, Universe &u);
```

---

## Universe.cpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #include "CelestialBody.hpp"
4 #include "Universe.hpp"
5
6 Universe::Universe(unsigned int num_planets, float radius) {
7     n = num_planets;
8     r = radius;
9 }
10
11 std::istream& operator>> (std::istream &input, Universe &u) {
12     input >> u.n;
13     input >> u.r;
14
15     for (unsigned int i = 0; i < u.n; i++) {
16         CelestialBody* c = new CelestialBody(u.r);
17         std::cin >> *c;
18         u.c_bodies.push_back(c);
19     }
20
21     return input;
22 }
23
24 std::ostream& operator<< (std::ostream &output, Universe &u) {
25     output << "Number of Planets: " << std::to_string(u.n) << std::endl;
26     output << "Radius: " << std::to_string(u.r) << std::endl;
27     for (CelestialBody *c : u.c_bodies) {
28         output << *c << std::endl;
29     }
30 }
```

```
30     return output;
31 }
32
33 void Universe::draw(sf::RenderTarget& target, sf::RenderStates states) const {
34     for (size_t i = 0; i < c_bodies.size(); i++) {
35         target.draw(*c_bodies[i]);
36     }
37 }
38
39 Universe::~Universe() {}
```

---

## main.cpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #include "Universe.hpp"
4 #include "CelestialBody.hpp"
5
6
7 int main(int argc, char* argv[]) {
8     Universe u;
9     std::cin >> u;
10
11     sf::RenderWindow w(sf::VideoMode(600, 600), "PS3a");
12     sf::Image background;
13
14     if (!background.loadFromFile("background.jpg")) {
15         return -1;
16     }
17
18     sf::Texture text;
19     text.loadFromImage(background);
20
21     sf::Sprite sprite;
22     sprite.setTexture(text);
23
24
25     while (w.isOpen()) {
26         sf::Event event;
27
28         while (w.pollEvent(event)) {
29             if (event.type == sf::Event::Closed)
30                 w.close();
31             w.draw(sprite);
32             w.draw(u);
33             w.display();
34         }
35     }
36 }
```

---

## Makefile

---

```
1
2 CC= g++
3 CFLAGS= --std=c++17 -Wall -Werror -pedantic
4 DEPS = CelestialBody.hpp Universe.hpp
5 LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
6
7 all: NBody
8
9 %.o: %.cpp $(DEPS)
10     $(CC) $(CFLAGS) -c $<
11
12 NBody: Universe.o CelestialBody.o main.o
13     $(CC) Universe.o CelestialBody.o main.o -o NBody $(LIB)
14
15 lint:
16     cpplint --filter=--runtime/references,-build/c++11 --root=. *.cpp *.hpp
17
18 clean:
19     rm *.o
20     rm NBody
21     rm *.gch
```

---

## PS3b: NBody Simulation (static)

### PROJECT SUMMARY AND WHAT I LEARNED

This assignment continues off of ps3a, adding animation and physics to the universe simulation. It uses the same files and classes as before and just appends to them. This is done by having the user input two new arguments into the command line: the total time and the time quantum ( $\Delta t$ ). Using these two values, the simulation then displays the orientation and movement of the universe up until that time, using increments of  $\Delta t$ . I also elected to use this project for my PSXa project. During my first go-around, I ran out of time despite knowing that I was missing a few features. So, in my second version, I reformatted the universe output in the command line to match the table format provided in the .txt files and printed the universe upon terminating the program in the main file. I also turned the vector of CelestialBody pointers in the Universe.hpp/.cpp file to a vector of CelestialBody smart pointers.

### KEY ALGORITHMS, DATA STRUCTURES AND/OR OO DESIGNS

I created an accelerate function inside of the CelestialBody class that handles all of the force, velocity, and acceleration calculations. This function is then called inside my step() function inside of Universe.hpp. It takes the time and a CelestialBody object as parameters, performs the calculations, and then updates the xvelocity and yvelocity of the celestial body. I also elected to update the elapsed time on the screen in my main function. I did this by creating a text object and drawing it to the window like a sprite.

### SCREENSHOT OF COMPILED PROGRAM



## CelestialBody.hpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #pragma once
4
5 #include <iostream>
6 #include <string>
7 #include <fstream>
8 #include <vector>
9 #include <SFML/Graphics.hpp>
10 #include <SFML/Window.hpp>
11
12 class CelestialBody: public sf::Drawable {
13 private:
14     void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;
15     double universe_radius, window_radius, xpos, ypos, xvel, yvel, mass;
16     std::string img_name;
17     sf::Image image;
18     sf::Sprite sprite;
19     sf::Texture texture;
20
21 public:
22     friend std::istream& operator>> (std::istream &input, CelestialBody &CB);
23     friend std::ostream& operator<< (std::ostream &output, CelestialBody &CB);
24
25     void accelerate(CelestialBody &, double time);
26     void updatePosition(double time);
27
28     explicit CelestialBody(double universe_radius, double window_radius = 300);
29 };
30
31 std::istream& operator>> (std::istream &input, CelestialBody &CB);
32 std::ostream& operator<< (std::ostream &output, CelestialBody &CB);
```

---

## CelestialBody.cpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #include "CelestialBody.hpp"
4 #include <cmath>
5
6 void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states)
7 const {
8     target.draw(sprite);
9 }
10
11 CelestialBody::CelestialBody(double universe_radius, double window_radius)
12 : universe_radius(universe_radius), window_radius(window_radius) {
13 }
14
15 std::istream& operator>> (std::istream &input, CelestialBody &CB) {
16     input >> CB.xpos >> CB.ypos;
17     input >> CB.xvel >> CB.yvel;
18     input >> CB.mass >> CB.img_name;
19
20     if (!CB.image.loadFromFile(CB.img_name)) {
21         throw std::ios_base::failure("Couldn't load image");
22     }
23
24     CB.texture.loadFromImage(CB.image);
25
26     CB.sprite.setTexture(CB.texture);
27 }
```

```

28     double x = (1 + CB.xpos/CB.universe_radius) * CB.window_radius;
29     double y = ((1-CB.ypos/CB.universe_radius) * CB.window_radius);
30     CB.sprite.setPosition(sf::Vector2f(x, y));
31
32     return input;
33 }
34
35
36 void CelestialBody::updatePosition(double time) {
37     xpos = xpos + xvel*time;
38     ypos = ypos + yvel*time;
39
40     double x = (1 +xpos/universe_radius) * window_radius;
41     double y = ((1-ypos/universe_radius) * window_radius);
42     sprite.setPosition(sf::Vector2f(x, y));
43 }
44
45 void CelestialBody::accelerate(CelestialBody &CB, double time) {
46     double dx = CB.xpos - xpos;
47     double dy = CB.ypos - ypos;
48     double r2 = dx * dx + dy * dy;
49     double r = sqrt(r2);
50
51     if (r == 0) {
52         std::cout << "fdsfsd" << std::endl;
53     }
54     double force = ((6.67e-11)*CB.mass*mass) / r2;
55
56     double forceX = force * (dx/r);
57     double forceY = force * (dy/r);
58
59     double accX = forceX / mass;
60     double accY = forceY / mass;
61
62     xvel = xvel + accX*time;
63     yvel = yvel + accY*time;
64 }
65
66 std::ostream& operator<< (std::ostream &output, CelestialBody &CB) {
67     output << " " << CB.xpos << " " << CB.ypos << " " << CB.xvel <<
68     " " << CB.yvel << " " << CB.mass << " " << CB.img_name;
69
70     return output;
71 }

```

---

## Universe.hpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #pragma once
4
5 #include <iostream>
6 #include <string>
7 #include <fstream>
8 #include <memory>
9 #include <vector>
10 #include <SFML/Graphics.hpp>
11 #include <SFML/Window.hpp>
12 #include "CelestialBody.hpp"
13
14 class Universe : public sf::Drawable {
15     private:
16         unsigned int n;
17         float r;
18         std::vector<std::shared_ptr<CelestialBody>> c_bodies;
19         void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;
20
21     public:
22         explicit Universe(unsigned int num_planets = 0, float radius = 0);
23
24         friend std::istream& operator>> (std::istream &input, Universe &u);
25         friend std::ostream& operator<< (std::ostream &output, Universe &u);
26
27         void step(double time);
28
29         ~Universe();
30 };
31
32 std::istream& operator>> (std::istream &input, Universe &u);
33 std::ostream& operator<< (std::ostream &output, Universe &u);
```

---

## Universe.cpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #include "CelestialBody.hpp"
4 #include "Universe.hpp"
5
6 Universe::Universe(unsigned int num_planets, float radius) {
7     n = num_planets;
8     r = radius;
9 }
10
11 std::istream& operator>> (std::istream &input, Universe &u) {
12     input >> u.n;
13     input >> u.r;
14
15     for (unsigned int i = 0; i < u.n; i++) {
16         std::shared_ptr<CelestialBody> c = std::make_shared<CelestialBody>(u.r);
17         std::cin >> *c;
18         u.c_bodies.push_back(c);
19     }
20
21     return input;
22 }
23
24
25 std::ostream& operator<< (std::ostream &output, Universe &u) {
26     output << "Number of Planets: " << u.n << std::endl;
```



```

27     output << "Radius: " << u.r << std::endl;
28     for (auto c : u.c_bodies) {
29         output << *c << std::endl;
30     }
31     return output;
32 }
33
34 void Universe::draw(sf::RenderTarget& target, sf::RenderStates states) const {
35     for (size_t i = 0; i < c_bodies.size(); i++) {
36         target.draw(*c_bodies[i]);
37     }
38 }
39 void Universe::step(double time) {
40     for (auto c : c_bodies) {
41         for (size_t i = 0; i < c_bodies.size(); i++) {
42             if (c != c_bodies[i]) {
43                 c->accelerate(*c_bodies[i], time);
44             }
45         }
46     }
47     for (auto c : c_bodies) {
48         c->updatePosition(time);
49     }
50 }
51
52 Universe::~Universe() {}

```

---

## main.cpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #include "Universe.hpp"
4 #include "CelestialBody.hpp"
5
6
7 int main(int argc, char* argv[]) {
8     std::string t = argv[1];
9     std::string delta_t = argv[2];
10
11     double t_double = std::stod(t);
12     double td_double = std::stod(delta_t);
13     double start = 0;
14
15     std::cout << td_double << std::endl;
16
17     Universe u;
18     std::cin >> u;
19
20     sf::RenderWindow w(sf::VideoMode(600, 600), "PS3a");
21     sf::Image background;
22
23     if (!background.loadFromFile("background.jpg")) {
24         return -1;
25     }
26
27     sf::Texture text;
28     sf::Font font;
29     font.loadFromFile("font.ttf");
30
31     sf::Text time_;
32
33     time_.setFont(font);
34
35     time_.setCharacterSize(30);
36
37     text.loadFromImage(background);
38
39     sf::Sprite sprite;
40     sprite.setTexture(text);
41
42
43     while (w.isOpen()) {
44         sf::Event event;
45
46         while (w.pollEvent(event)) {
47             if (event.type == sf::Event::Closed)
48                 w.close();
49         }
50         w.clear();
51         w.draw(sprite);
52         time_.setString("Elapsed Time (seconds): " + std::to_string(start));
53         w.draw(time_);
54         u.step(td_double);
55         w.draw(u);
56         w.display();
57
58         start += td_double;
59
60         if (start >= t_double) {
61             break;
62         }
63     }
```

```
64     std::cout << u << std::endl;
65 }
```

---

## Makefile

---

```
1
2 CC= g++
3 CFLAGS= --std=c++17 -Wall -Werror -pedantic
4 DEPS = CelestialBody.hpp Universe.hpp
5 LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
6
7 all: NBody
8
9 %.o: %.cpp $(DEPS)
10     $(CC) $(CFLAGS) -c $<
11
12 NBody: Universe.o CelestialBody.o main.o
13     $(CC) Universe.o CelestialBody.o main.o -o NBody $(LIB)
14
15 lint:
16     cpplint --filter=--runtime/references,-build/c++11 --root=. *.cpp *.hpp
17
18 clean:
19     rm *.o
20     rm NBody
21     rm *.gch
```

---

## PS4a: CircularBuffer

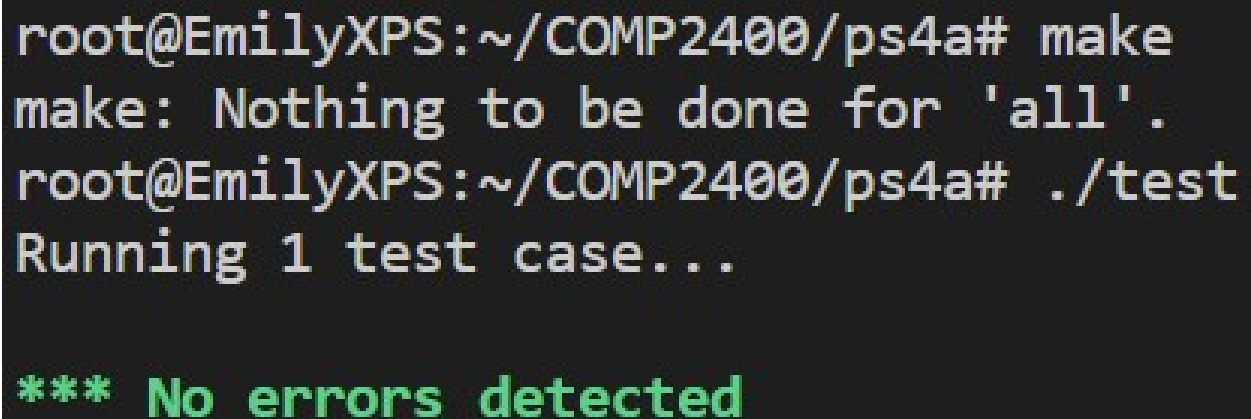
### PROJECT SUMMARY AND WHAT I LEARNED

This project was by far the simplest of the semester. In fact, it almost felt too straightforward compared to previous assignments, so much so that I joined office hours just to make sure I wasn't missing an extra page of the assignment or misunderstanding the directions. That being said, this project implements a template class called CircularBuffer that serves as the base class structure for building a guitar in ps4b. It has two member variables, a queue and an int. The CircularBuffer queue has a max capacity, the int, that dictates how much it can hold. There is also a test file that uses Boost again to test each function.

### KEY ALGORITHMS, DATA STRUCTURES AND/OR OO DESIGNS

This was the first project built with a template. I made the CircularBuffer a template class. I built the class using a queue as my main data function. The functions required for implementation, size(), isEmpty(), isFull(), enqueue(), dequeue(), and peek() made a queue an easy choice over a vector because C++ has built in functions for many of these functions already. The project also throws exceptions if the queue tries to push back additional T objects over capacity, or if it tries to pop while the queue is empty. It throws invalid argument and runtime error exceptions. In test.cpp, I created an empty CircularBuffer object with a size of three initially. I first ensured that isEmpty was working by using BOOST CHECK. I then called BOOST REQUIRE THROW to make sure that a runtime error was thrown when calling peek() and deque() on this empty CB. Afterwards, I enqueued my first number, and called BOOST REQUIRE NO THROW. I then enqueued two more values, confirmed its size using BOOST CHECK EQUAL and then finally concluded with checking that the size of the CB was indeed full via isFull().

### SCREENSHOT OF COMPILED PROGRAM



```
root@EmilyXPS:~/COMP2400/ps4a# make
make: Nothing to be done for 'all'.
root@EmilyXPS:~/COMP2400/ps4a# ./test
Running 1 test case...

*** No errors detected
```

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #pragma once
4
5 #include <string>
6 #include <iostream>
7 #include <queue>
8 #include <exception>
9
10 template<typename T>
11 class CircularBuffer {
12 public:
13     explicit CircularBuffer(size_t c);
14     size_t size() const;
15     bool isEmpty() const;
16     bool isFull() const;
17     void enqueue(T item);
18     T dequeue();
19     T peek() const;
20 private:
21     size_t capacity;
22     std::queue<T> q;
23 };
24
25 template<typename T>
26 CircularBuffer<T>::CircularBuffer(size_t c) : capacity(c), q() {
27     if (capacity < 1) {
28         throw std::invalid_argument("capacity less than 1");
29     }
30 }
31
32 template<typename T>
33 size_t CircularBuffer<T>::size() const {
34     return q.size();
35 }
36
37 template<typename T>
38 bool CircularBuffer<T>::isEmpty() const {
39     return q.empty();
40 }
41
42 template<typename T>
43 bool CircularBuffer<T>::isFull() const {
44     if (q.size() == capacity) {
45         return true;
46     }
47     return false;
48 }
49
50 template<typename T>
51 void CircularBuffer<T>::enqueue(T item) {
52     if (isFull()) {
53         throw std::runtime_error("buffer is already at capacity");
54     }
55     q.emplace(item);
56 }
57
58 template<typename T>
59 T CircularBuffer<T>::dequeue() {
60     if (isEmpty()) {
61         throw std::runtime_error("buffer is empty");
62     }
63     T t = q.front();
```

```

64     q.pop();
65     return t;
66 }
67
68 template<typename T>
69 T CircularBuffer<T>::peek() const {
70     if (isEmpty()) {
71         throw std::runtime_error("buffer is empty");
72     }
73     return q.front();
74 }

```

---

test.cpp

---

```

1  // "Copyright 2022 Emily Sheehan"
2
3  #include <queue>
4
5  #include "CircularBuffer.hpp"
6
7  #define BOOST_TEST_DYN_LINK
8  #define BOOST_TEST_MODULE Main
9  #include <boost/test/unit_test.hpp>
10 #include <boost/test/tools/output_test_stream.hpp>
11
12 using boost::test_tools::output_test_stream;
13
14 output_test_stream output;
15
16 BOOST_AUTO_TEST_CASE(test) {
17     CircularBuffer<int> cb(3);
18     BOOST_CHECK(cb.isEmpty());
19     BOOST_REQUIRE_THROW(cb.peek(), std::runtime_error);
20     BOOST_REQUIRE_THROW(cb.dequeue(), std::runtime_error);
21     BOOST_REQUIRE_NO_THROW(cb.enqueue(123));
22     cb.enqueue(2);
23     cb.enqueue(3);
24     BOOST_CHECK_EQUAL(cb.size(), 3);
25     BOOST_CHECK(cb.isFull());
26     BOOST_REQUIRE_THROW(cb.enqueue(4), std::runtime_error);
27 }

```

---

## Makefile

---

```
1 CC= g++
2 CFLAGS= --std=c++17 -Wall -Werror -pedantic -g
3 DEPS = CircularBuffer.hpp
4 LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
5
6 all: test
7
8 %.o: %.cpp $(DEPS)
9     $(CC) $(CFLAGS) -c $<
10
11 test: test.o
12     $(CC) $^ -o test $(LIB)
13
14 lint:
15     cpplint --filter=--runtime/references,-build/c++11 --root=. *.cpp *.hpp
16
17 clean:
18     rm *.o
19     rm test
20     rm *.gch
```

---

## PS4b: KSGuitarSim

### PROJECT SUMMARY AND WHAT I LEARNED

This project completes project ps4a, adding the full functionality of the keytar. It simulates the instrument through the pressing of keys on the keyboard that correspond to different notes. It uses the Karplus-Strong algorithm to create the guitar plucking sounds. In addition to the graphics library from SFML that we've used many times previously, we also took advantage of the library's audio package which was a new feature. Structurally, this project is composed of our CircularBuffer template class from ps4a, a StringSound class, and a KSGuitarSim file that runs the program.

### KEY ALGORITHMS, DATA STRUCTURES AND/OR OO DESIGNS

The heart of this project is the Karplus-Strong algorithm. Per Wikipedia, the "Karplus-Strong string synthesis is a method of physical modelling synthesis that loops a short waveform through a filtered delay line to simulate the sound of a hammered or plucked string or some types of percussion." We modeled this in our StringSound class which defines the functions pluck() and tick() to calculate the rhythm and correct sounds to play. The tick() function specifically find the average of the first note enqueued and the last enqueued and then appends this the CircularBuffer (to be played next).

### SCREENSHOT OF COMPILED PROGRAM

```
root@EmilyXPS:~/COMP2400/ps4b# ./test
Running 3 test cases...

*** No errors detected
root@EmilyXPS:~/COMP2400/ps4b#
```



CircularBuffer.hpp (see ps4a)  
StringSound.hpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #include <iostream>
4 #include <vector>
5 #include <cmath>
6 #include <random>
7 #include <SFML/Audio.hpp>
8
9 #include "CircularBuffer.hpp"
10
11 #pragma once
12
13 class StringSound {
14 public:
15     explicit StringSound(double frequency);
16     StringSound(double frequency, unsigned int seed);
17     explicit StringSound(std::vector<sf::Int16> init);
18     StringSound(const StringSound &obj) = delete;
19     void pluck();
20     void tick();
21     sf::Int16 sample() const;
22     size_t time() const;
23
24 private:
25     size_t tics;
26     CircularBuffer<sf::Int16> *cb;
27     std::mt19937 gen;
28 };
```

---

StringSound.cpp

---

```
1 // "Copyright 2022 Emily Sheehan"
2
3 #include <chrono>
4 #include <vector>
5 #include "StringSound.hpp"
6
7
8 StringSound::StringSound(double frequency) {
9     unsigned int seed = std::chrono::system_clock::now()
10         .time_since_epoch().count();
11     gen.seed(seed);
12     int buffer_size = ceil(44100/frequency);
13     this->tics = 0;
14     this->cb = new CircularBuffer<sf::Int16>(buffer_size);
15     for (int x=0; x < buffer_size; x++) {
16         cb->enqueue(0);
17     }
18 }
19 StringSound::StringSound(double frequency, unsigned int seed)
20     : StringSound(frequency) {
21     gen.seed(seed);
22 }
23
24 StringSound::StringSound(std::vector<sf::Int16> init) {
25     unsigned int seed = std::chrono::system_clock::now()
26         .time_since_epoch().count();
27     gen.seed(seed);
28     this->cb = new CircularBuffer<sf::Int16>(init.size());
29     this->tics = 0;
30     for (std::size_t i = 0; i < init.size(); i++) {
```

```

31         this -> cb -> enqueue(init.at(i));
32     }
33 }
34 void StringSound::pluck() {
35     int max = 32767;
36     int min = -32768;
37     std::uniform_int_distribution<int> dist(min, max);
38
39     cb->empty();
40     while (!cb->isFull()) {
41         int num = dist(gen);
42         cb -> enqueue(num);
43     }
44 }
45
46 void StringSound::tick() {
47     tics++;
48     sf::Int16 deleted = cb->dequeue();
49     sf::Int16 first = cb->peek();
50     sf::Int16 avg = .996*((deleted + first)/2);
51     cb->enqueue(avg);
52 }
53
54 sf::Int16 StringSound::sample() const {
55     return this->cb->peek();
56 }
57
58 size_t StringSound::time() const {
59     return tics;
60 }

```

---

```
1  /*
2   Copyright 2015 Fred Martin,
3   Y. Rykalova, 2020
4   J. Daly 2022
5  */
6
7  #include <math.h>
8  #include <limits.h>
9
10 #include <iostream>
11 #include <string>
12 #include <exception>
13 #include <stdexcept>
14 #include <vector>
15
16 #include <SFML/Graphics.hpp>
17 #include <SFML/System.hpp>
18 #include <SFML/Audio.hpp>
19 #include <SFML/Window.hpp>
20
21 #include "CircularBuffer.hpp"
22 #include "StringSound.hpp"
23
24 #define TOTAL 400
25 #define SAMPLES_PER_SEC 44100
26
27 std::vector<sf::Int16> makeSamples(StringSound &gs) {
28     std::vector<sf::Int16> samples;
29     gs.pluck();
30     int duration = 8;
31     int i;
32     for (i = 0; i < SAMPLES_PER_SEC * duration; i++) {
33         gs.tick();
34         samples.push_back(gs.sample());
35     }
36
37     return samples;
38 }
39
40 int main() {
41     sf::RenderWindow window(sf::VideoMode(300, 200),
42                             "SFML Plucked String Sound Lite");
43     sf::Event event;
44     double freq;
45     std::vector<std::vector<sf::Int16>> samples;
46
47     freq = TOTAL;
48     StringSound ss(freq);
49     sf::Sound sound1;
50     sf::SoundBuffer buf1;
51
52     int index;
53     std::string keys = "q2we4r5ty7u8i9op-=[zxdcfvgbnjmk,.;/' ";
54
55     int vals[] = {16, 28, 22, 4, 30, 17, 31, 19, 24, 33,
56                  20, 34, 8, 35, 14, 15, 56, 46, 55, 25,
57                  23, 3, 2, 5, 21, 6, 1, 13, 9, 12, 10,
58                  49, 50, 48, 52, 51, 57};
59
60     std::vector<int> numbers;
61
62     for (int i = 0; i < 37; i++) {
63         std::vector<sf::Int16> s;
```

```

64     numbers.push_back(vals[i]);
65     freq = 440 * (pow(2.0, ((24 - i) / 12.0)));
66     ss = StringSound(freq);
67     try {
68         s = makeSamples(ss);
69     }
70     catch (const std::exception &e) {
71         std::cout << "error in make samples" << std::endl;
72     }
73     samples.push_back(s);
74 }
75
76 while (window.isOpen()) {
77     while (window.pollEvent(event)) {
78         if (event.type == sf::Event::Closed) {
79             window.close();
80         }
81
82         if (event.type == sf::Event::KeyPressed) {
83             index = std::find(numbers.begin(), numbers.end(),
84                               event.key.code) -
85                               numbers.begin();
86             if (index < 37) {
87                 if (!buf1.loadFromSamples(&(samples[index][0]),
88                                           samples[index].size(), 2, SAMPLES_PER_SEC)) {
89                     throw std::runtime_error(
90                         "sf::SoundBuffer: failed to load from samples");
91                 }
92                 sound1.setBuffer(buf1);
93                 sound1.play();
94             }
95         }
96     }
97     return 0;
98 }
99

```

---

## Makefile

---

```
1 CC= g++
2 CFLAGS= --std=c++17 -Wall -Werror -pedantic -g
3 DEPS = CircularBuffer.hpp StringSound.hpp
4 LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
5
6 all: test KSGuitarSim
7
8 %.o: %.cpp $(DEPS)
9     $(CC) $(CFLAGS) -c $<
10
11
12 KSGuitarSim: KSGuitarSim.o StringSound.o
13     $(CC) $^ -o KSGuitarSim $(LIB)
14
15 test: test.o StringSound.o
16     $(CC) $^ -o test $(LIB)
17
18 lint:
19     cpplint --filter=--runtime/references,-build/c++11 --root=. *.cpp *.hpp
20
21 clean:
22
23     rm *.o
24     rm KSGuitarSim
25     rm test
26     rm *.gch
```

---

## PS5: RandWriter

### PROJECT SUMMARY AND WHAT I LEARNED

This project obtains a body of text, analyzes it over a fixed number of characters and a specific letter, calculates the probability of certain letters in the text, and then generates nonsense text based off of the model it's built. It contains four files, including a test file. The project is based off of the RandWriter class which takes into account Markov's order and frequency principles. It implements Markov chaining in order to create the nonsense text, which was a totally foreign concept to me prior to this assignment. Essentially, Markov chaining works by picking a starting sequence (or single character), noting the preceding letter, then searching for that letter, notes its preceding letter, then searches for that letter, and so on... It then generates text by using the probability of the occurrences of the preceding letters/strings. My program works by running this set of logic over a selected text by a k- number of characters. There are also tests, once again using Boost, to ensure the correctness of my generated text.

### KEY ALGORITHMS, DATA STRUCTURES AND/OR OO DESIGNS

Along with the required class functions, I added four member variables to my header file: an int "num" to represent the order in the Markov model, a map to hold my kgrams, a string to represent the alphabet/letters used, and a string to represent the words in the input. In my constructor, instead of looping through character by character and appending to a temporary string, I used substrings over a range. This implementation allowed for looping back to the beginning of the string which the other did not, thereby raising errors. Secondly, the most challenging part of this assignment for me was creating tests for the generate() function. I talked with Dr. Daly who then pointed me to some simple statistics. I created ranges for how many of each letter should appear in a 1000 character string that I generated and tested each character against the values that it should fall between.

### SCREENSHOT OF COMPILED PROGRAM

```
root@EmilyXPS:~/COMP2400/ps5# ./test
Running 4 test cases...

*** No errors detected
root@EmilyXPS:~/COMP2400/ps5#
```

## RandWriter.hpp

---

```
1 // Copyright 2022 Emily Sheehan
2
3 #include <algorithm>
4 #include <iostream>
5 #include <map>
6 #include <string>
7
8 #pragma once
9
10 class RandWriter {
11 public:
12     RandWriter(std::string text, int k);
13     int orderK() const;
14     int freq(std::string kgram) const;
15     int freq(std::string kgram, char c) const;
16     char kRand(std::string kgram);
17     std::string generate(std::string kgram, int T);
18     friend std::ostream& operator<< (std::ostream &o, RandWriter &rw);
19
20 private:
21     int num;
22     std::map<std::string, int> kgrams;
23     std::string letters_used;
24     std::string words;
25 };
```

---

## RandWriter.cpp

---

```
1 // Copyright 2022 Emily Sheehan
2
3 #include "RandWriter.hpp"
4 #include <bits/stdc++.h>
5
6 #include <string>
7 #include <algorithm>
8 #include <map>
9 #include <utility>
10 #include <stdexcept>
11 #include <vector>
12 #include <ctime>
13 #include <cmath>
14 #include <chrono>
15 #include <random>
16
17 RandWriter::RandWriter(std::string text, int k) {
18     if (k < 0) {
19         throw std::invalid_argument("k must be at least 0");
20     }
21     if (static_cast<int>(text.size()) < k) {
22         throw std::invalid_argument("text length must be larger than k");
23     }
24
25     srand(time(NULL));
26     num = k;
27     std::string words = text;
28     for (unsigned int i = 0; i < text.size(); i++) {
29         size_t pos = letters_used.find(text[i]);
30         if (std::string::npos == pos) {
31             letters_used.push_back(text[i]);
32         }
33     }
34 }
```

```

35     for (unsigned int i = 0; i < text.size(); i++) {
36         std::string temp;
37         if ((i+k) > text.size()) {
38             temp = text.substr(i) + text.substr(0, k - text.size() + i);
39         } else {
40             temp = text.substr(i, k);
41         }
42         char c = text[(i+k) % text.size()];
43         kgrams[temp]++;
44         kgrams[temp+c]++;
45     }
46 }
47
48 int RandWriter::orderK() const {
49     return num;
50 }
51
52 int RandWriter::freq(std::string kgram) const {
53     int klen = kgram.length();
54
55     if (klen != num) {
56         throw std::runtime_error("kgram is not length k");
57     }
58     auto it = kgrams.find(kgram);
59
60     if (it == kgrams.end()) {
61         return 0;
62     }
63
64     return it->second;
65 }
66
67 int RandWriter::freq(std::string kgram, char c) const {
68     int klen = kgram.length();
69     if (klen != num) {
70         throw std::runtime_error("kgram is not length k");
71     }
72
73     kgram.push_back(c);
74     auto it = kgrams.find(kgram);
75
76     if (it == kgrams.end()) {
77         return 0;
78     }
79
80     return it->second;
81 }
82
83 char RandWriter::kRand(std::string kgram) {
84     auto it = kgrams.find(kgram);
85
86     int klen = std::count_if(kgram.begin(),
87                             kgram.end(),
88                             [](char c) {
89                                 return true;
90                             });
91     // int klen = count(kgram.begin(), kgram.end(), [](int i) { return i++; });
92
93     if (klen != num) {
94         throw std::runtime_error("kgram is not of length k");
95     }
96     if (it == kgrams.end()) {
97         throw std::runtime_error("no such kgram");
98     }
99     std::mt19937 gen;

```



```

100     unsigned int seed = std::chrono::system_clock::now()
101         .time_since_epoch().count();
102     gen.seed(seed);
103     int max = 32767;
104     int min = -32768;
105     std::uniform_int_distribution<int> dist(min, max);
106     int q = dist(gen);
107
108     int ran = q % freq(kgram);
109     double test_freq = 0;
110     double ran2 = static_cast<double>(ran) / freq(kgram);
111     double prev = 0;
112
113     for (unsigned int i= 0; i< (unsigned)letters_used.length(); i++) {
114         int f = freq(kgram, letters_used[i]);
115
116         test_freq = static_cast<double>(f) / freq(kgram);
117         if (ran2 < test_freq + prev && test_freq != 0) {
118             return letters_used[i];
119         }
120         prev = prev + test_freq;
121     }
122     return ' ';
123 }
124
125 std::string RandWriter::generate(std::string kgram, int L) {
126     int klen = kgram.length();
127
128     if (klen != num) {
129         throw std::runtime_error("kgram not of length k");
130     }
131
132     std::string final_string = "";
133     char return_char;
134     final_string += kgram;
135
136     for (int i= 0; i< (L - num); i++) {
137         std::string sub = final_string.substr(i, num);
138         return_char = kRand(sub);
139         final_string.push_back(return_char);
140     }
141     return final_string;
142 }
143
144 std::ostream &operator<<(std::ostream &o, RandWriter &rw) {
145     o << "Original: " << rw.words << std::endl;
146     o << "Order: " << rw.num << std::endl;
147     o << "Alphabet: " << rw.letters_used << std::endl;
148
149     std::map<std::string, int>::iterator it;
150     for (it = rw.kgrams.begin(); it != rw.kgrams.end(); it++) {
151         o << it->first << " " << it->second << std::endl;
152     }
153     return o;
154 }

```

---

## TextWriter.cpp

---

```
1 // Copyright Emily Sheehan 2022
2
3 #include <iostream>
4 #include <string>
5 #include <sstream>
6 #include "RandWriter.hpp"
7
8 int main(int argc, char* argv[]) {
9     std::string temp;
10    std::string original;
11    std::string transformed;
12
13    std::string warning = "Wrong number of arguments - "
14    " format should be ./TextWriter L K < <text file>";
15    if (argc != 3) {
16        std::cout << warning << std::endl;
17    }
18
19    int k = atoi(argv[1]);
20    int L = atoi(argv[2]);
21
22
23    while (std::cin >> transformed) {
24        original += " " + transformed;
25        transformed = "";
26    }
27
28
29    for (int i = 0; i < k; i++) {
30        temp.push_back(original[i]);
31    }
32    std::cout << "After applying Markov transformation: "
33    << RandWriter(original, k).generate(temp, L) << std::endl;
34 }
```

---

## Makefile

---

```
1 CC = g++
2 CFLAGS = -g -Wall -Werror -std=c++17 -pedantic
3 DEPS = RandWriter.hpp
4 LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
5
6 all: TextWriter test
7
8 %.o: %.cpp $(DEPS)
9     $(CC) $(CFLAGS) -c $<
10
11 test: test.o RandWriter.o
12     $(CC) $^ -o test $(LIB)
13
14 TextWriter: TextWriter.o RandWriter.o
15     $(CC) $^ -o TextWriter $(LIB)
16
17 lint:
18     cpplint --filter=--runtime/references,-build/c++11 --root=. *.cpp *.hpp
19
20
21 clean:
22     rm *.o
23     rm TextWriter
24     rm test
25     rm lint
```

---

## PS7: Kronos Log Parsing

### PROJECT SUMMARY AND WHAT I LEARNED

This final project focuses on parsing Kronos logs with Regular expressions. We verified the cleanliness of the logs and outputted whether the boot-up of the device was successful or not by writing to a file. This project was not lengthy (only one main file) but it incorporated two new concepts– regex and writing to a file. Regex proved to be very useful and efficient while writing to a file was exciting because I can imagine implementing it in a real world situation/applying it to a job.

### KEY ALGORITHMS, DATA STRUCTURES AND/OR OO DESIGNS

As mentioned, this project uses one file, kronos.cpp. I basically pulled the time calculations from the .cpp file provided with the project guide. I used a while loop, also suggested in the stdin boost.cpp file, to loop through line by line of the log file. I then checked to see whether the line currently read in matched the starting or ending time, or neither. Depending on this boolean, I printed to the report file. These are the regex statements that I used:

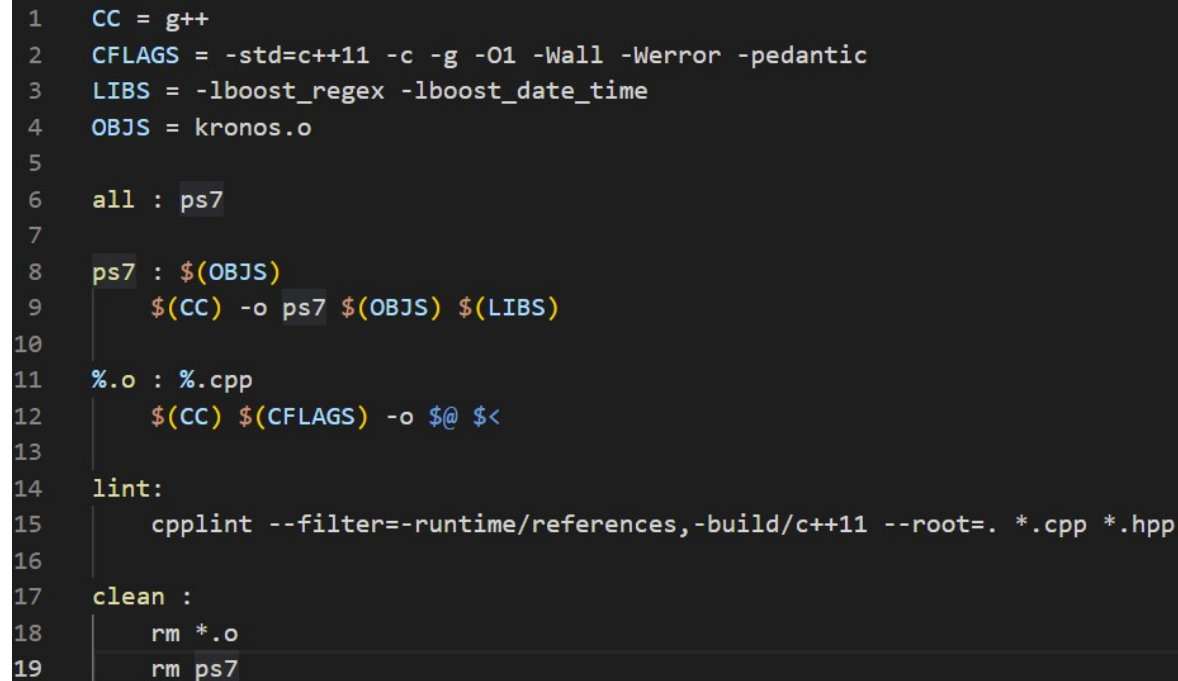
I used two which indicate the starting time (eb) and time of completion (ed).

---

```
1  regex eb("[0-9]{4})-([0-9]{1,2})-([0-9]{1,2})"
2      " ([0-9]{2}):([0-9]{2}):([0-9]{2})"
3      + std::string(".*log.c.166.*) server started"));
4  regex ed("[0-9]{4})-([0-9]{1,2})-([0-9]{1,2})"
5      " ([0-9]{2}):([0-9]{2}):([0-9]{2})"
6      + std::string(".*oejs.AbstractConnector:"
7      "Started SelectChannelConnector@0.0.0.0:9080.*"));
```

---

### SCREENSHOT OF COMPILED PROGRAM



```
1  CC = g++
2  CFLAGS = -std=c++11 -c -g -O1 -Wall -Werror -pedantic
3  LIBS = -lboost_regex -lboost_date_time
4  OBJS = kronos.o
5
6  all : ps7
7
8  ps7 : $(OBJS)
9      $(CC) -o ps7 $(OBJS) $(LIBS)
10
11 %.o : %.cpp
12     $(CC) $(CFLAGS) -o $@ $<
13
14 lint:
15     cpplint --filter=-runtime/references,-build/c++11 --root=. *.cpp *.hpp
16
17 clean :
18     rm *.o
19     rm ps7
```

---

```

1 // Copyright 2022 Emily Sheehan
2 #include <cstdlib>
3 #include <iostream>
4 #include <string>
5 #include <fstream>
6 #include <vector>
7 #include <boost/regex.hpp>
8
9
10
11 #include "boost/date_time/gregorian/gregorian.hpp"
12 #include "boost/date_time/posix_time/posix_time.hpp"
13
14 using boost::regex;
15 using boost::gregorian::date;
16 using boost::gregorian::from_simple_string;
17 using boost::posix_time::time_from_string;
18 using boost::posix_time::ptime;
19 using boost::posix_time::time_duration;
20
21
22 int main(int argc, char *argv[]) {
23     std::ifstream iF(argv[1], std::ifstream::in);
24     std::string name = std::string(argv[1]) + ".rpt";
25     std::string oFName(name);
26     std::ofstream oF;
27     boost::smatch m;
28     std::string line;
29     int bootStart = 0;
30     int bootComplete = 0;
31     ptime t1;
32
33     if (argc != 2) {
34         std::cout << "Wrong # of args" << std::endl;;
35         return -1;
36     }
37
38     if (!iF.is_open()) {
39         std::cout << "Can't open file." << std::endl;;
40         return -1;
41     }
42
43     oF.open(oFName.c_str());
44
45     regex eb("[0-9]{4})-([0-9]{1,2})-([0-9]{1,2})"
46             " ([0-9]{2}):([0-9]{2}):([0-9]{2})"
47             + std::string(".*log.c.166.*"));
48     regex ed("[0-9]{4})-([0-9]{1,2})-([0-9]{1,2})"
49             " ([0-9]{2}):([0-9]{2}):([0-9]{2})"
50             + std::string(".*oejs.AbstractConnector:"
51             "Started SelectChannelConnector@0.0.0.0:9080.*"));
52
53     int lineNum = 1;
54     bool startToBoot = false;
55
56     while (getline(iF, line)) {
57         if (regex_match(line, m, eb)) {
58             if (startToBoot) {
59                 oF << "**** Incomplete boot **** \n" << std::endl;;
60             }
61             startToBoot = true;
62
63             date d_(from_simple_string(m[0]));

```

```

64         time_duration td1(atoi(m[4].str().c_str()),
65                             atoi(m[5].str().c_str()),
66                             atoi(m[6].str().c_str()));
67     ptime temp(d_, td1);
68     t1 = temp;
69
70     std::string x1 = std::to_string(lineNum) + "(" + argv[1] + "):";
71     std::string x2 = m[1] + "-" + m[2] + "-" + m[3] + " ";
72     std::string x3 = m[4] + ":" + m[5] + ":" + m[6] + " ";
73     oF << "=== Device boot ===" << std::endl;;
74     oF << x1;
75     oF << x2;
76     oF << x3;
77     oF << "Boot Start" << std::endl;;
78     bootStart++;
79 } else if (regex_match(line, m, ed)) {
80     if (startToBoot) {
81         startToBoot = false;
82
83         date d_(from_simple_string(m[0]));
84         time_duration td2(atoi(m[4].str().c_str()),
85                             atoi(m[5].str().c_str()),
86                             atoi(m[6].str().c_str()));
87         ptime temp(d_, td2);
88         time_duration td = temp - t1;
89
90         std::string x1 = std::to_string(lineNum) + "(" + argv[1] + "):";
91         std::string x2 = m[1] + "-" + m[2] + "-" + m[3] + " ";
92         std::string x3 = m[4] + ":" + m[5] + ":" + m[6] + " ";
93
94         oF << x1;
95         oF << x2;
96         oF << x3;
97         oF << "Boot Completed\n";
98         oF << "\tBoot Time: ";
99
100        std::string time1 = std::to_string(td.total_milliseconds())
101        + "ms \n\n";
102        oF << time1;
103        bootComplete++;
104    } else {
105        oF << "**** Unexpected boot ****\n\n";
106    }
107 }
108
109     lineNum++;
110 }
111
112     oF << "Device Boot Report\n\n"
113     << "InTouch log file: " << argv[1] << std::endl
114     << "Lines Scanned: " << lineNum - 1 << std::endl << std::endl
115     << "Device boot count: initiated: " << bootStart << ", completed: "
116     << bootComplete << std::endl << std::endl;
117
118     return 0;
119 }

```

---

## Makefile

```

1 CC = g++
2 CFLAGS = -std=c++11 -c -g -O1 -Wall -Werror -pedantic
3 LIBS = -lboost_regex -lboost_date_time
4 OBJS = kronos.o
5

```

```
6 all : ps7
7
8 ps7 : $(OBJJS)
9      $(CC) -o ps7 $(OBJJS) $(LIBS)
10
11 %.o : %.cpp
12      $(CC) $(CFLAGS) -o $@ $<
13
14 lint:
15      cpplint --filter=--runtime/references,-build/c++11 --root=. *.cpp *.hpp
16
17 clean :
18      rm *.o
19      rm ps7
```

---