

Computing IV: Section 201

Project Portfolio

John Hua

Spring 2022

Contents

1	PS0: Hello World with SFML	6
1.1	Overview	6
1.2	Key Concepts	6
1.3	Learnings	6
1.4	Output	7
1.5	PS0 Code	8
1.5.1	main.cpp	8
2	PS1a: Linear Feedback Shift Register	11
2.1	Overview	11
2.2	Key Concepts	11
2.3	Learnings	11
2.4	Output	12
2.5	PS1a Makefile	13
2.5.1	Makefile	13
2.6	PS1a FibLFSR Source	14
2.6.1	FibLFSR.cpp	14
2.7	PS1a FibLFSR Header	17
2.7.1	FibLFSR.h	17
2.8	PS1a Boost Test Source	18
2.8.1	test.cpp	18
3	PS1b: PhotoMagic	20
3.1	Overview	20
3.2	Key Concepts	20
3.3	Learnings	20
3.4	Output	21
3.5	PS1b Makefile	22
3.5.1	Makefile	22
3.6	PS1b FibLFSR Source	23
3.6.1	FibLFSR.cpp	23
3.7	PS1b FibLFSR Header	26
3.7.1	FibLFSR.h	26
3.8	PS1b PhotoMagic Source	28
3.8.1	PhotoMagic.cpp	28

4	PS2a: Static N-Body Simulation	31
4.1	Overview	31
4.2	Key Concepts	31
4.3	Learnings	31
4.4	Output	32
4.5	PS2a Makefile	33
4.5.1	Makefile	33
4.6	PS2a Main Source	34
4.6.1	main.cpp	34
4.7	PS2a CelestialBody Source	37
4.7.1	CelestialBody.cpp	37
4.8	PS2a CelestialBody Header	40
4.8.1	CelestialBody.h	40
4.9	PS2a Universe Header	42
4.9.1	Universe.h	42
5	PS2b: N-Body Simulation	45
5.1	Overview	45
5.2	Key Concepts	45
5.3	Learnings	45
5.4	Output	46
5.5	PS2b Makefile	47
5.5.1	Makefile	47
5.6	PS2b Main Source	48
5.6.1	main.cpp	48
5.7	PS2b CelestialBody Source	54
5.7.1	CelestialBody.cpp	54
5.8	PS2b CelestialBody Header	58
5.8.1	CelestialBody.h	58
5.9	PS2b Universe Header	61
5.9.1	Universe.h	61
6	PS3: Triangle Fractal	64
6.1	Overview	64
6.2	Key Concepts	64
6.3	Learnings	64
6.4	Output	65
6.5	PS3 Makefile	66
6.5.1	Makefile	66
6.6	PS3 TFractal Source	67

6.6.1	TFractal.cpp	67
6.7	PS3 Triangle Source	69
6.7.1	Triangle.cpp	69
6.8	PS3 Triangle Header	71
6.8.1	Triangle.h	71
7	PS4a: CircularBuffer	73
7.1	Overview	73
7.2	Key Concepts	73
7.3	Learnings	73
7.4	Output	74
7.5	PS4a Makefile	75
7.5.1	Makefile	75
7.6	PS4a Main Source	76
7.6.1	main.cpp	76
7.7	PS4a CircularBuffer Source	78
7.7.1	CircularBuffer.cpp	78
7.8	PS4a CircularBuffer Header	81
7.8.1	CircularBuffer.h	81
7.9	PS4a Boost Source	83
7.9.1	test.cpp	83
8	PS4b: String Sound	85
8.1	Overview	85
8.2	Key Concepts	85
8.3	Learnings	85
8.4	Output	86
8.5	PS4b Makefile	87
8.5.1	Makefile	87
8.6	PS4b Main Source	89
8.6.1	main.cpp	89
8.7	PS4b CircularBuffer Source	91
8.7.1	CircularBuffer.cpp	91
8.8	PS4b CircularBuffer Header	94
8.8.1	CircularBuffer.h	94
8.9	PS4b StringSound Source	96
8.9.1	StringSound.cpp	96
8.10	PS4b StringSound Header	99
8.10.1	StringSound.h	99
8.11	PS4b Boost Source	101

8.11.1	test.cpp	101
9	PS5: DNA Alignment	103
9.1	Overview	103
9.2	Key Concepts	103
9.3	Learnings	103
9.4	Output	104
9.5	PS5 Makefile	105
9.5.1	Makefile	105
9.6	PS5 Main Source	106
9.6.1	main.cpp	106
9.7	PS5 EDistance Source	108
9.7.1	EDistance.cpp	108
9.8	PS5 EDistance Header	112
9.8.1	EDistance.h	112
10	PS6: Random Writer	114
10.1	Overview	114
10.2	Key Concepts	114
10.3	Learnings	114
10.4	Output	115
10.5	PS6 Makefile	116
10.5.1	Makefile	116
10.6	PS6 Main Source	117
10.6.1	main.cpp	117
10.7	PS6 RandWriter Source	119
10.7.1	RandWriter.cpp	119
10.8	PS6 RandWriter Header	124
10.8.1	RandWriter.h	124
10.9	PS6 Boost Source	126
10.9.1	test.cpp	126
11	PS7: Kronos Time Parsing	128
11.1	Overview	128
11.2	Key Concepts	128
11.3	Learnings	128
11.4	Output	129
11.5	PS7 Makefile	130
11.5.1	Makefile	130
11.6	PS7 Main Source	131

11.6.1	main.cpp	131
--------	--------------------	-----

1 PS0: Hello World with SFML

1.1 Overview

PS0 was an introduction to the SFML library that we would be using for a majority of our assignments throughout the semester. The assignment set students up with the build environment to code using SFML. I had chosen to install Ubuntu using WSL (short-handed for Windows sub-system for Linux) as my build environment. The rest of PS0 was installing the necessary libraries and extending the demo SFML code to ensure that everything was working.

1.2 Key Concepts

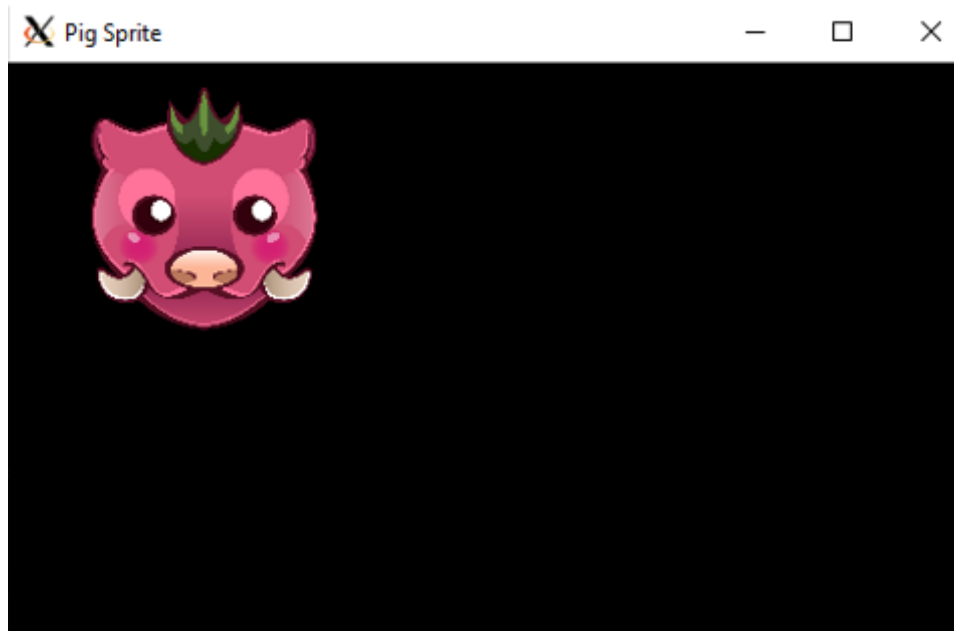
The sample SFML code given introduced us to the event handler that outputted the display of sprites and among other things to the screen. We were also given a link to the official documentation and asked to extend the demo code to get our feet wet with SFML. No algorithms or data structures were necessarily introduced in PS0 but rather important procedures and protocol that we had to follow for the rest of our assignments for the remainder of the semester.

1.3 Learnings

As an introductory assignment, I learned important concepts such as setting up a proper build environment for future assignments as well as how to experiment with SFML. In addition to setting up the necessary libraries and extending the demo code, we also learned how to compile using the proper flags and how to properly submit assignments using the 'tar' command to zip up our files to send to the grader.

1.4 Output

Figure 1: Sample output of PS0 running with a sprite displayed



1.5 PS0 Code

1.5.1 main.cpp

```
1 /*
2  * Created by: John Hua
3  * Computing IV, Instructor: Professor Daly
4  * Assignment: ps0, due: 1/24/2022
5  *
6  *
7  * Comments: This program is an introductory to
      using the SFML libraries. It implements
8  * detecting keypresses to change the position of
      the sprite as well using the getSize()
9  * function to detect when the sprite's position is
      out bounds of the window region.
10 *
11 *
12 *
13 */
14 #include <iostream>
15 using namespace std;
16 #include <SFML/Graphics.hpp>
17
18
19
20 int main()
21 {
22     sf::RenderWindow window(sf::VideoMode(500, 300),
        "Pig Sprite");
23     sf::Texture pig;
24     if (!pig.loadFromFile("sprite.png")) {
25         cout << "Unable to load piggie sprite." <<
            endl;
26         return 0;
27     }
28     sf::Texture octo;
29     if (!octo.loadFromFile("octo.png")) {
30         cout << "Unable to load octo sprite." <<
            endl;
```

```

31         return 0;
32     }
33
34     sf::Sprite piggieSprite;
35     piggieSprite.setTexture(pig);
36     piggieSprite.setScale(0.2, 0.2);
37
38     sf::Sprite octoSprite;
39     octoSprite.setTexture(octo);
40     octoSprite.setScale(0.2, 0.2);
41
42     while (window.isOpen())
43     {
44
45         sf::Vector2f pos = piggieSprite.getPosition
46             ();
47         sf::Event event;
48         while (window.pollEvent(event))
49         {
50             if (event.type == sf::Event::Closed)
51                 window.close();
52         }
53
54         if (sf::Keyboard::isKeyPressed(sf::Keyboard
55             ::Right)) {
56             piggieSprite.move(1, 0);
57         }
58         else if (sf::Keyboard::isKeyPressed(sf::
59             Keyboard::Left)) {
60             piggieSprite.move(-1, 0);
61         }
62         else if (sf::Keyboard::isKeyPressed(sf::
63             Keyboard::Up)) {
64             piggieSprite.move(0, -1);
65         }
66         else if (sf::Keyboard::isKeyPressed(sf::
67             Keyboard::Down)) {
68             piggieSprite.move(0, 1);
69         }
70     }

```

```
66         window.clear();
67         if ((pos.x < 0 || pos.x > window.getSize().x
68             ) ||
69             (pos.y < 0 || pos.y > window.getSize().y)
70             ) {
71             window.draw(octoSprite);
72         } else {
73             window.draw(piggieSprite);
74         }
75         window.display();
76     }
77 }
```

2 PS1a: Linear Feedback Shift Register

2.1 Overview

PS1a is a two-part assignment where we build a linear feedback register to produce pseudo-random bits; one of many methods or tools used in cryptography. The linear feedback shift register works by performing a left shift of a string of bits and the shifted off bit gets XOR'd by tap positions in the bit string with the final result replacing the vacated bit on the end. We were left to our own interpretation on how to implement this linear feedback shift register using the given API consisting of the constructor and the step and generate function.

2.2 Key Concepts

Key concepts of PS1a included learning how to implement the linear feedback shift register in our own way that we felt would sufficiently meet the requirements. This set the precedent for the majority of the class where future assignments are left to our own interpretation on how to properly implement an algorithm or data structure. In addition to implementing the linear feedback shift register, we were also required to learn how to install and implement the Boost testing framework that we would also be using for the majority of our future assignments. This framework would test our program and methods; an important concept in determining if a software design or program is fit for use.

2.3 Learnings

For the linear feedback shift register, I implemented it by manipulating a string using methods in the string class such as 'pop' and 'push_back' to manipulate individual bits in the string. I also learned how to install the Boost development framework to test my code. This assignment also shook off some rust from Computing III on the implementation of classes, objects, constructors, etc. which I found fairly informative as a refresher. We were also tasked with implementing exceptions in the case that incorrect input was passed to the constructor, which was also tested in the Boost test program.

2.4 Output

Figure 2: Sample output of PS1a running with LFSR test string: "1011011000110110" testing the functions 'generate' and 'step'

```
jhua@DESKTOP-75DLUJV:~/computing_IV/ps1a$ ./main
1011011000110110      0
0110110001101100      0
1101100011011000      0
1011000110110000      1
0110001101100001      1
1100011011000011      0
1000110110000110      0
0001101100001100      1
0011011000011001      1
0110110000110011      0
1101100001100110      236
```

2.5 PS1a Makefile

2.5.1 Makefile

```
1 CC = g++
2 CFLAGS = -Wall -Werror -pedantic --std=c++14
3 LIBS = -lboost_unit_test_framework
4 DEPS = FibLFSR.h
5
6 #%.o: %.cpp $(DEPS)
7 #     $(CC) $(CFLAGS) -c $<
8
9 all: main ps1a
10
11 ps1a: FibLFSR.o test.o
12     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
13
14 main: main.o FibLFSR.o
15     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
16
17 FibLFSR.o: FibLFSR.cpp FibLFSR.h
18     $(CC) $(CFLAGS) -c FibLFSR.cpp $(LIBS)
19
20 main.o: main.cpp FibLFSR.h
21     $(CC) $(CFLAGS) -c main.cpp $(LIBS)
22
23 test.o: FibLFSR.h
24     $(CC) $(CFLAGS) -c test.cpp $(LIBS)
25
26 clean:
27     rm *.o ps1a
```

2.6 PS1a FibLFSR Source

2.6.1 FibLFSR.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps1a>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <1.31.2022>*/
6  /*Sources Of Help: <Daly's Comp4 Discord Server /
   Boost Documentation>*/
7  /*
   *****
   */
8
9
10 #include "FibLFSR.h"
11 using namespace std;
12
13
14 FibLFSR::FibLFSR(string seed) {
15     if (seed.length() != 16) {
16         throw std::out_of_range("Out of range:
           Invalid seed input.");
17     } else {
18         _register = seed;
19     }
20 }
21
22 string FibLFSR::get15Bits() {
23
24     _register.pop_back();
25     return _register;
26
27 }
28
29 int FibLFSR::getRegSize() {
30
31     return _register.length();
32 }
```

```

33 }
34
35
36 int FibLFSR::step() {
37     string temp;
38
39     int newbit =
40         _register.at(0) ^
41         _register.at(2) ^
42         _register.at(3) ^
43         _register.at(5);
44
45     for (unsigned int i = 0; i < _register.length()
46         -1; i++) {
47         temp.push_back(_register.at(i + 1));
48     }
49
50     temp.push_back(newbit+48);
51     _register = temp;
52
53     return newbit;
54 }
55
56 int FibLFSR::generate(int k) {
57     int genBit = 0;
58
59     for (int j = 0; j < k; j++) {
60         genBit = genBit*2 + step();
61     }
62
63     return genBit;
64 }
65 }
66
67
68
69 ostream& operator<< (ostream& output, const FibLFSR&
70     fiblfsr) {
71     output << fiblfsr._register;

```



```

71     return output;
72 }
73
74 /*int main() {
75
76     FibLFSR fiblfsr("1011011000110110");
77
78     // cout << fiblfsr << " " << fiblfsr.step() <<
        endl;
79     cout << fiblfsr << "      " << fiblfsr.generate(9)
        << endl;
80
81     return 0;
82 }
83 */

```

2.7 PS1a FibLFSR Header

2.7.1 FibLFSR.h

```
1 #ifndef FibLFSR_H
2 #define FibLFSR_H
3 #include <iostream>
4 #include <string>
5 using namespace std;
6
7
8 class FibLFSR {
9     public:
10
11         FibLFSR(string seed); // constructor to
            create LFSR with the given initial seed
12
13         int step(); // int step(); // simulate one
            step and return the new bit as 0 or 1
14
15         int generate(int k); // int generate(int k);
            // simulate k steps and return k-bit
            integer
16
17         string get15Bits(); // Returns the first 15-
            bits of a register. For unit testing
            purposes.
18
19         int getRegSize(); // Returns the full
            register size (16-bits). For unit testing
            purposes.
20
21         friend ostream& operator<<(ostream& out,
            const FibLFSR& fiblfsr);
22
23     private:
24         string _register;
25 };
26
27 #endif
```

2.8 PS1a Boost Test Source

2.8.1 test.cpp

```
1 // Dr. Rykalova
2 // test.cpp for PS1a
3 // updated 1/31/2020
4
5 #include <iostream>
6 #include <string>
7
8 #include "FibLFSR.h"
9
10 #define BOOST_TEST_DYN_LINK
11 #define BOOST_TEST_MODULE Main
12 #include <boost/test/unit_test.hpp>
13
14 BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
15
16     FibLFSR l("1011011000110110");
17     BOOST_REQUIRE(l.step() == 0);
18     BOOST_REQUIRE(l.step() == 0);
19     BOOST_REQUIRE(l.step() == 0);
20     BOOST_REQUIRE(l.step() == 1);
21     BOOST_REQUIRE(l.step() == 1);
22     BOOST_REQUIRE(l.step() == 0);
23     BOOST_REQUIRE(l.step() == 0);
24     BOOST_REQUIRE(l.step() == 1);
25
26     FibLFSR l2("1011011000110110");
27     BOOST_REQUIRE(l2.generate(9) == 51);
28 }
29
30 /*An more in-depth test for the step function where
   it checks
31 that the register bits has been correctly shifted
   once to the
32 left after one call to the step function. The test
   checks the
33 first 15-bits and ignores the last bit.
```

```

34 */
35
36 BOOST_AUTO_TEST_CASE(checkLeftShift) {
37     FibLFSR a("0110101010001100");
38     a.step();
39     BOOST_REQUIRE(a.get15Bits() == "110101010001100");
40 }
41
42 /*Checks the number of bits in the seed that's
    passed to the
43 constructor is equal to 16, otherwise throw an
    exception. This
44 does not check whether the seed is valid meaning the
    register
45 contains only 0's and 1's.*/
46
47 BOOST_AUTO_TEST_CASE(checkExcpt) {
48     BOOST_REQUIRE_THROW(FibLFSR abc("011"), std::
        out_of_range);
49 }

```

3 PS1b: PhotoMagic

3.1 Overview

PS1b is the second part of the two-part assignment dealing the linear feedback shift registers and cryptography. This particular assignment required us to use the linear feedback register we made back in the first part to now encode/decode an image and display it onto a second window and save it onto the drive. Sample starter code was given in the file 'pixels.cpp' where it would create a photographic negative image of the upper top-left of corner of the image. We would then use this starter code and implement it with out linear feedback shift register to encode each individual pixel with a different RGB value in row major form to transform it.

3.2 Key Concepts

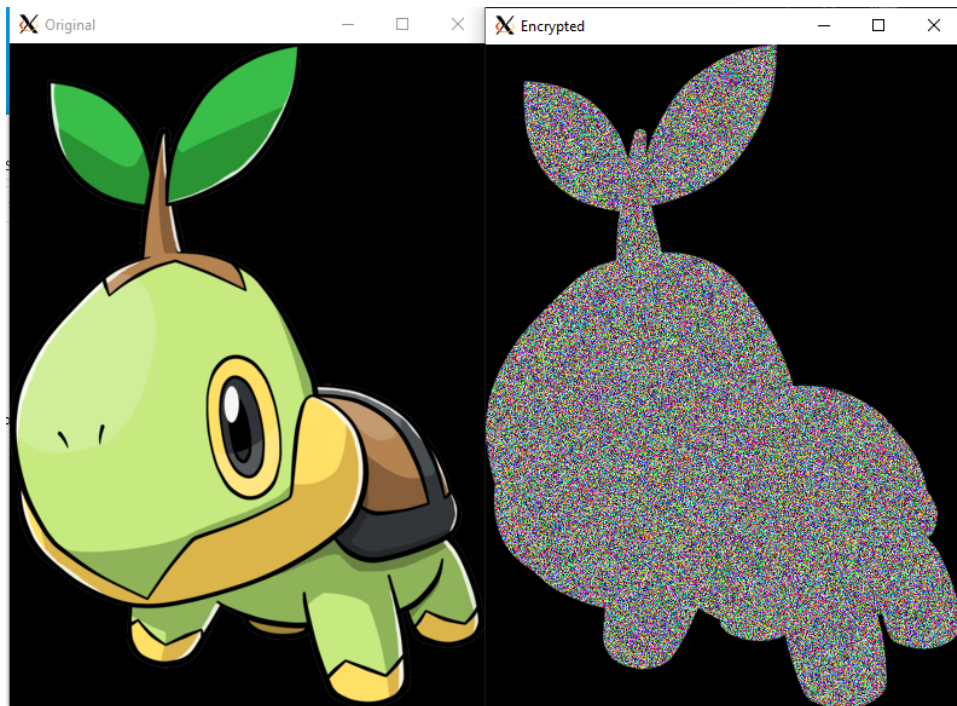
PS1b is a follow-up from PS1a where we now use our newly created linear feedback shift register and incorporate it together with the SFML library to encode and decode an image. One of the important key concepts was that each individual pixel in an image had an RGB value and could be individually manipulated. The other key concept is that copying the image is memory intensive so the alternative was to create a sprite that loaded the original to the first render window, call the transform function to encrypt the image, then create another sprite to load that encrypted image onto a second render window.

3.3 Learnings

In this assignment, I learned how to incorporate my linear feedback shift register with the SFML library. I also learned a bit of cryptography since this assignment provided an introductory to that sort of particular discipline. The sample starter code provided showed that each individual pixel could be manipulated, and this could open up hundreds of other possibilities to transform images in different ways. The other takeaway from this assignment was that copying images in SFML is extremely memory intensive, so this hinted at using other methods to display the second encoded/decoded image instead of simply copying it.

3.4 Output

Figure 3: Sample image of an image being encoded from the code running



3.5 PS1b Makefile

3.5.1 Makefile

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

3.6 PS1b FibLFSR Source

3.6.1 FibLFSR.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps1a>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <1.31.2022>*/
6  /*Sources Of Help: <Daly's Comp4 Discord Server /
   Boost Documentation>*/
7  /*
   ****
   */
8
9
10 #include "FibLFSR.h"
11 using namespace std;
12
13
14 FibLFSR::FibLFSR(string seed) {
15     if (seed.length() != 16) {
16         throw std::out_of_range("Out of range:
           Invalid seed input.");
17     } else {
18         _register = seed;
19     }
20 }
21
22 string FibLFSR::get15Bits() {
23
24     _register.pop_back();
25     return _register;
26
27 }
28
29 int FibLFSR::getRegSize() {
30
31     return _register.length();
32 }
```



```

33 }
34
35
36 int FibLFSR::step() {
37     string temp;
38
39     int newbit =
40         _register.at(0) ^
41         _register.at(2) ^
42         _register.at(3) ^
43         _register.at(5);
44
45     for (unsigned int i = 0; i < _register.length()
46         -1; i++) {
47         temp.push_back(_register.at(i + 1));
48     }
49
50     temp.push_back(newbit+48);
51     _register = temp;
52
53     return newbit;
54 }
55
56 int FibLFSR::generate(int k) {
57     int genBit = 0;
58
59     for (int j = 0; j < k; j++) {
60         genBit = genBit*2 + step();
61     }
62
63     return genBit;
64 }
65 }
66
67
68
69 ostream& operator<< (ostream& output, const FibLFSR&
70     fiblfsr) {
71     output << fiblfsr._register;

```

```

71     return output;
72 }
73
74 /*int main() {
75
76     FibLFSR fiblfsr("1011011000110110");
77
78     // cout << fiblfsr << " " << fiblfsr.step() <<
        endl;
79     cout << fiblfsr << "      " << fiblfsr.generate(9)
        << endl;
80
81     return 0;
82 }
83 */

```

3.7 PS1b FibLFSR Header

3.7.1 FibLFSR.h

```
1 #ifndef FibLFSR_H
2 #define FibLFSR_H
3 #include <iostream>
4 #include <string>
5
6 #include <SFML/System.hpp>
7 #include <SFML/Window.hpp>
8 #include <SFML/Graphics.hpp>
9
10 using namespace std;
11
12
13 class FibLFSR {
14     public:
15
16         FibLFSR(string seed); // constructor to
            create LFSR with the given initial seed
17
18         int step(); // int step(); // simulate one
            step and return the new bit as 0 or 1
19
20         int generate(int k); // int generate(int k);
            // simulate k steps and return k-bit
            integer
21
22         string get15Bits(); // Returns the first 15-
            bits of a register. For unit testing
            purposes.
23
24         int getRegSize(); // Returns the full
            register size (16-bits). For unit testing
            purposes.
25
26         friend ostream& operator<<(ostream& out,
            const FibLFSR& fiblfsr);
27
```

```
28     private:
29         string _register;
30 };
31
32 #endif
```

3.8 PS1b PhotoMagic Source

3.8.1 PhotoMagic.cpp

```
1 /*
   *****

2  *Name: <John Hua>
3  *Course name: <COMP.2040>
4  *Assignment: <PS1b>
5  *Instructor's name: <Dr. James Daly>
6  *Date: <2.7.2022>
7  *Sources Of Help: <Dr Daly's Discord Server>
8  *****
   */

9
10 #include <SFML/System.hpp>
11 #include <SFML/Window.hpp>
12 #include <SFML/Graphics.hpp>
13
14 #include "FibLFSR.h"
15
16 // transform function
17 void transform(sf::Image& image, FibLFSR*);
18
19 int main(int argc, char* argv[])
20 {
21     sf::Image image;
22     if (!image.loadFromFile(argv[1]))
23         return -1;
24
25     FibLFSR fiblfsr(argv[3]);
26
27     sf::Vector2u size = image.getSize();
28
29     sf::RenderWindow window1(sf::VideoMode(size.x,
        size.y), "Original");
30     sf::RenderWindow window2(sf::VideoMode(size.x,
        size.y), "Encrypted");
31
```

```

32 // Load original image to window1
33 sf::Texture texture;
34 texture.loadFromImage(image);
35
36 sf::Sprite sprite;
37 sprite.setTexture(texture);
38
39 // Encode image
40 transform(image, &fblfsrc);
41
42 // Load encoded image to window2 from new sprite
   to avoid copying
43 sf::Texture texture_2;
44 texture_2.loadFromImage(image);
45
46 sf::Sprite sprite_2;
47 sprite_2.setTexture(texture_2);
48
49 while (window1.isOpen() || window2.isOpen())
50 {
51     sf::Event event;
52     while (window1.pollEvent(event))
53     {
54         if (event.type == sf::Event::Closed)
55             window1.close();
56     }
57
58     while (window2.pollEvent(event))
59     {
60         if (event.type == sf::Event::Closed)
61             window2.close();
62     }
63
64     window1.clear(sf::Color::Black);
65     window1.draw(sprite);
66     window1.display();
67
68     window2.clear(sf::Color::Black);
69     window2.draw(sprite_2);
70     window2.display();

```

```

71     }
72
73     // fredm: saving a PNG segfaults for me, though
       it does properly
74     // write the file
75     if (!image.saveToFile(argv[2]))
76         return -1;
77
78     return 0;
79 }
80
81 void transform(sf::Image& image, FibLFSR* fiblfsr) {
82
83     // p is a pixelimage.getPixel(x, y);
84     sf::Color p;
85
86     for (unsigned int x = 0; x < image.getSize().x;
          x++) {
87         for (unsigned int y = 0; y < image.getSize()
          .y; y++) {
88             p = image.getPixel(x, y);
89             p.r = p.r ^ fiblfsr->generate(10);
90             p.g = p.g ^ fiblfsr->generate(8);
91             p.b = p.b ^ fiblfsr->generate(13);
92             image.setPixel(x, y, p);
93         }
94     }
95 }

```

4 PS2a: Static N-Body Simulation

4.1 Overview

PS2a is the first part of a two-part assignment where it provides an introductory to creating a simulation of the solar system. PS2a required us to first create a static simulation of the universe or solar system rather, where we tasked with using SFML to first draw the five celestial bodies of the solar system onto a display window in their respective positions according to a text file. The latter part of the assignment would require us to simulate the planets moving or orbiting around the sun.

4.2 Key Concepts

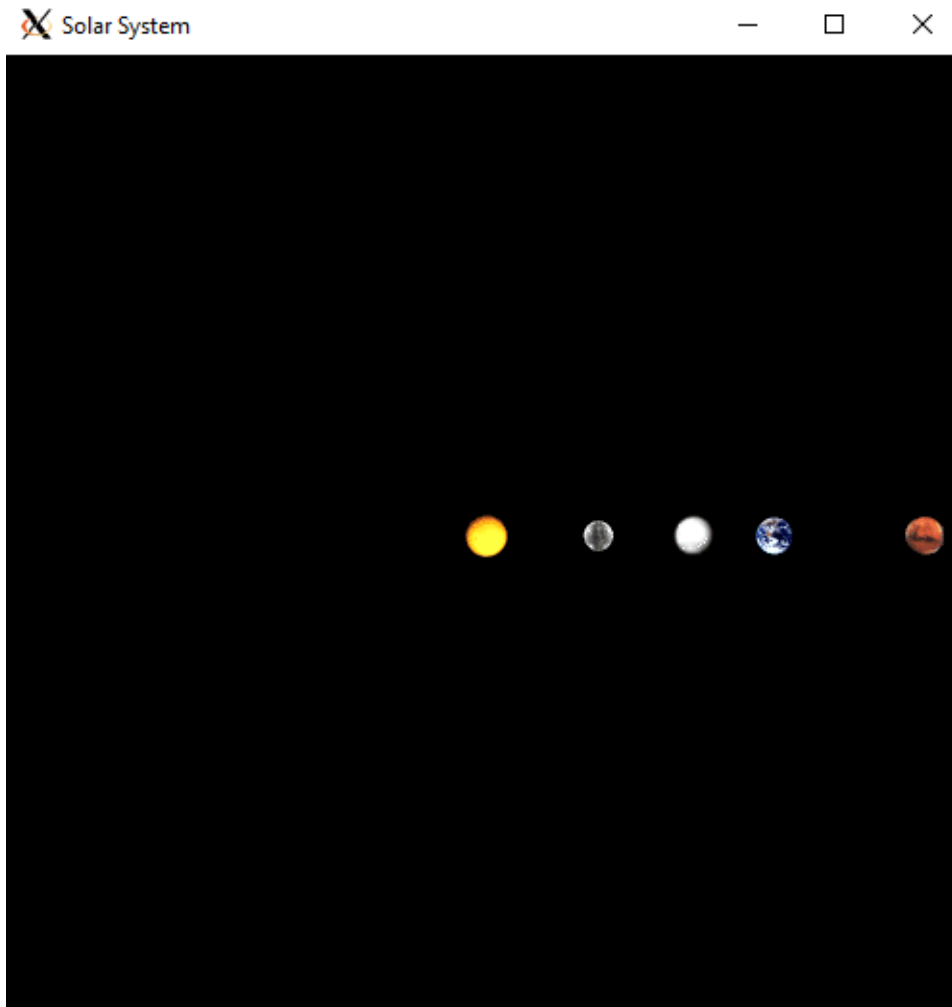
The assignment required using a vector of pointers to objects. A majority of the time was spent figuring out how to pass the data structure around to different functions to manipulate the member variables, as well as overloading the input stream operator correctly to read in the initialization parameters. This assignment provided important key concepts such as implementing smart pointers, manipulating data structures, and how to properly access the vector of pointers to transform the celestial body contained within the universe to transform its properties such as its position, acceleration, velocity, etc.

4.3 Learnings

For this static n-body simulation assignment, I learned how to create a vector of pointers to objects and instantiate them properly as required in the assignment. This required a bit of work and time to correctly instantiate each celestial body properly and to ensure they were in their correct positions in the static display of the simulation. This was an important first step since the universe would contain the vector of pointers to objects, as we would later on the latter assignment be required to incorporate a step function within the universe so SFML can move each celestial body in each consecutive frame. Another important thing I learned was how to receive input from a text file, as we would be reading in the positions, acceleration, and other properties through a given text file.

4.4 Output

Figure 4: Static simulation of the solar system in their starting positions



4.5 PS2a Makefile

4.5.1 Makefile

```
1 CC = g++
2 CFLAGS = -pedantic --std=c++14
3 LIBS = -lsfml-graphics -lsfml-window -lsfml-system
4 DEPS = CelestialBody.h
5
6 #%.o: %.cpp $(DEPS)
7 #     $(CC) $(CFLAGS) -c $<
8
9 all: NBody
10
11 NBody: main.o CelestialBody.o
12     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
13
14 main.o: main.cpp Universe.h
15     $(CC) $(CFLAGS) -c main.cpp $(LIBS)
16
17 CelestialBody.o: CelestialBody.cpp CelestialBody.h
18     $(CC) $(CFLAGS) -c CelestialBody.cpp $(LIBS)
19
20 clean:
21     rm *.o NBody
```

4.6 PS2a Main Source

4.6.1 main.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps2a>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <2.12.2022>*/
6  /*Sources Of Help: <Daly's Comp4 Discord Server /
   Boost Documentation>*/
7  /*
   *****
   */
8
9  #include <iostream>
10 #include <fstream>
11 #include <string>
12
13 #include "CelestialBody.h"
14 #include "Universe.h"
15
16 int main () {
17
18     /*Read in the first two lines of planets.txt (
       the number of celestial bodies then the
       universe radius*/
19     int num_body;
20     std::cin >> num_body;
21     std::cout << num_body << std::endl; // Testing
       if it read correctly
22
23     double universe_radius;
24     std::cin >> universe_radius;
25     std::cout << universe_radius<< std::endl;
26
27     /*Instantiate the Universe*/
28     Universe solar_system(universe_radius);
29
30     /*Create RenderWindow*/
```

```

31     sf::RenderWindow window(sf::VideoMode(500, 500),
        "Solar System");
32     sf::Vector2u window_size = window.getSize(); //
        Store render window x,y coordinates
33
34     /*Manually create CelestialBody objects*/
35     //CelestialBody sun(0, 0, 0, 0, 0, "sun.gif");
36     //CelestialBody earth(1.4960e+11, 0, 0, 0, 0, "
        earth.gif");
37     //CelestialBody mars(2.2790e+11, 0, 0, 0, 0, "
        mars.gif");
38     //CelestialBody mercury(5.7900e+10, 0, 0, 0, 0,
        "mercury.gif");
39
40     /*Have the Universe class instantiate new
        CelestialBody objects*/
41     //solar_system.body.push_back (new CelestialBody
        (2.2790e+11, 0, 0, 0, 0, "mars.gif"));
42     //solar_system.body.push_back (new CelestialBody
        (5.7900e+10, 0, 0, 0, 0, "mercury.gif"));
43     //solar_system.body.push_back (new CelestialBody
        (0.0000e+00, 0, 0, 0, 0, "sun.gif"));
44     //solar_system.body.push_back (new CelestialBody
        (1.0820e+11, 0, 0, 0, 0, "venus.gif"));
45
46
47     /*For loop to instantiate new celestial bodies,
        initialize the parameters from the overloaded
48     input stream operator, then load the image,
        texture, and sprite*/
49     for(int i = 0; i < num_body; i++) {
50         solar_system.body.push_back (new
            CelestialBody);
51         std::cin >> solar_system.body[i];
52         solar_system.body[i]->loadSprite();
53     }
54
55     while (window.isOpen())
56     {
57         sf::Event event;

```

```

58     while (window.pollEvent(event))
59     {
60         if (event.type == sf::Event::Closed)
61             window.close();
62     }
63
64     window.clear(sf::Color::Black);
65
66     /*Draw the sprites and position them onto
67     the viewport. Can maybe clean
68     this up later with a for-loop.*/
69
70     window.draw(*solar_system.body[0]);
71     window.draw(*solar_system.body[1]);
72     window.draw(*solar_system.body[2]);
73     window.draw(*solar_system.body[3]);
74     window.draw(*solar_system.body[4]);
75
76     solar_system.body[0]->renderToWindow(
77         window_size, universe_radius);
78     solar_system.body[1]->renderToWindow(
79         window_size, universe_radius);
80     solar_system.body[2]->renderToWindow(
81         window_size, universe_radius);
82     solar_system.body[3]->renderToWindow(
83         window_size, universe_radius);
84     solar_system.body[4]->renderToWindow(
85         window_size, universe_radius);
86     //solar_system.renderToWindow(solar_system.
87         body[0]); (NOT USED)
88     window.display();
89 }
90
91 return 0;
92 }

```

4.7 PS2a CelestialBody Source

4.7.1 CelestialBody.cpp

```
1 #include <iostream>
2 #include <string>
3
4 #include <SFML/System.hpp>
5 #include <SFML/Window.hpp>
6 #include <SFML/Graphics.hpp>
7
8 #include "CelestialBody.h"
9
10 CelestialBody::CelestialBody() { }
11
12 CelestialBody::CelestialBody(double xp, double yp,
13     double xv, double yv, double m, std::string img)
14 {
15     xpos = xp;
16     ypos = yp;
17     xvel = xv;
18     yvel = yv;
19     mass = m;
20     filename = img;
21 }
22
23
24 void CelestialBody::renderToWindow(const sf::
25     Vector2u& window_size, double universe_radius) {
26     /*Translate viewport to Cartesian coordinate
27     system*/
28     double xNew = (((universe_radius*2)/2) + xpos) *
29         (window_size.x/(universe_radius*2));
30     double yNew = (((universe_radius*2)/2) - ypos) *
31         (window_size.y/(universe_radius*2));
32
33     /*Gets bounding box of sprite and sets origin to
34     center by dividing width/height by 2*/
```

```

31     sf::FloatRect sprite_dimensions = sprite.
        getGlobalBounds();
32     sprite.setOrigin(sprite_dimensions.width/2,
        sprite_dimensions.height/2);
33
34     sprite.setPosition(xNew, yNew);
35
36 }
37
38 void CelestialBody::loadSprite() {
39
40     sf::Image image;
41
42     if (!image.loadFromFile(filename))
43         std::cout << "Cannot load image file." <<
            std::endl;
44
45     texture.loadFromImage(image);
46
47     sprite.setTexture(texture);
48
49 }
50
51 /*Mutator Functions*/
52 void CelestialBody::setXpos(double x) {
53
54     xpos = x;
55
56 }
57
58 void CelestialBody::setYpos(double y) {
59
60     ypos = y;
61
62 }
63
64 void CelestialBody::setXvel(double xv) {
65
66     xvel = xv;
67

```

```
68 }
69
70 void CelestialBody::setYvel(double yv) {
71
72     yvel = yv;
73
74 }
75
76 void CelestialBody::setMass(double m) {
77
78     mass = m;
79
80 }
81
82 void CelestialBody::setFileName(std::string file) {
83
84     filename = file;
85
86 }
```


4.8 PS2a CelestialBody Header

4.8.1 CelestialBody.h

```
1 #ifndef CelestialBody_H
2 #define CelestialBody_H
3 #include <iostream>
4 #include <string>
5
6 #include <SFML/System.hpp>
7 #include <SFML/Window.hpp>
8 #include <SFML/Graphics.hpp>
9
10
11 class CelestialBody : public sf::Drawable
12 {
13     public:
14         /*Constructors*/
15         CelestialBody();
16         CelestialBody(double xp, double yp, double
            xv, double yv, double m, std::string img)
            ;
17
18         /*Mutator Functions*/
19         void setXpos(double x);
20         void setYpos(double y);
21         void setXvel(double xv);
22         void setYvel(double yv);
23         void setMass(double m);
24         void setFileName(std::string file);
25
26         /*Load the image file and set the texture
            and sprite*/
27         void loadSprite();
28
29         /*Translate viewport to Cartesian
            coordinates and set sprite's origin (0,0)
            to be centered*/
30         void renderToWindow(const sf::Vector2u&
            window_size, double universe_radius);
```

```

31
32     private:
33         /*Private virtual void draw function*/
34         virtual void draw(sf::RenderTarget& target,
35                             sf::RenderStates states) const
36         {
37             target.draw(sprite);
38         }
39         /*Private member variables*/
40         double xpos;
41         double ypos;
42         double xvel;
43         double yvel;
44         double mass;
45         std::string filename;
46
47         /*Sprite and texture member variables*/
48         sf::Sprite sprite;
49         sf::Texture texture;
50 };
51
52 #endif

```

4.9 PS2a Universe Header

4.9.1 Universe.h

```
1 #ifndef Universe_H
2 #define Universe_H
3 #include <iostream>
4 #include <string>
5
6 #include <SFML/System.hpp>
7 #include <SFML/Window.hpp>
8 #include <SFML/Graphics.hpp>
9
10 using namespace std;
11
12
13 class Universe : public sf::Drawable
14 {
15     public:
16         /*Constructor(s)*/
17         Universe(double rad);
18
19         /*Vector of pointers to objects (
20          CelestialBody)*/
21         std::vector<CelestialBody*> body;
22
23         /*Overloaded input stream operator*/
24         friend istream& operator>>(std::istream&
25             input, CelestialBody *&body);
26
27     private:
28         virtual void draw(sf::RenderTarget& target,
29             sf::RenderStates states) const
30         {
31             target.draw(sprite);
32         }
33         double win_size;
34
35         double radius;
36         sf::Sprite sprite;
```

```

34         sf::Texture texture;
35     };
36
37
38     Universe::Universe(double rad) {
39         radius = rad;
40     }
41
42     /*Overloaded istream operator to take inputs. Passes
        the pointer(*) to object by reference(&) to
        avoid copying*/
43     istream& operator>>(istream& input, CelestialBody *&
        body) {
44
45         double _xpos;
46         double _ypos;
47         double _xvel;
48         double _yvel;
49         double _mass;
50         std::string _filename;
51
52         input >> _xpos >> _ypos >> _xvel >> _yvel >>
            _mass >> _filename;
53
54         body->setXpos(_xpos);
55         body->setYpos(_ypos);
56         body->setXvel(_xvel);
57         body->setYvel(_yvel);
58         body->setMass(_mass);
59         body->setFileName(_filename);
60
61         return input;
62     }
63
64     /*Passing a pointer to the object (NOT USED)*/
65
66     /*void Universe::renderToWindow(CelestialBody *&body
        )
67     {
68

```

```
69     body->sprite.setPosition(200, 200);
70
71 }
72 */
73
74 #endif
```

5 PS2b: N-Body Simulation

5.1 Overview

PS2b is the second part of the two-part assignment where we are now are tasked with simulating the movement of the solar system by manipulating the celestial bodies to orbit around the sun. The former assignment had us set-up a static display of the planets. This required us to extend our `CelestialBody` class with mutators so that our physics simulation can manipulate the velocities and positions of each celestial object.

5.2 Key Concepts

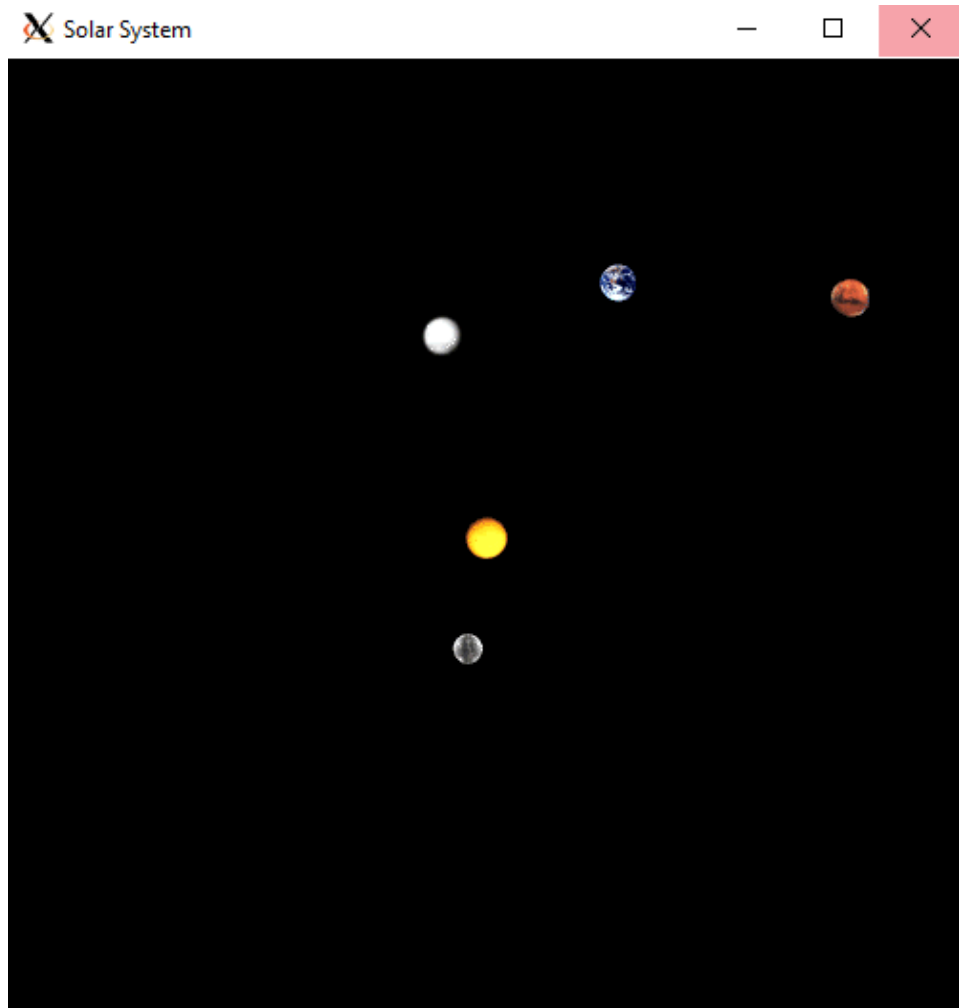
Important key concepts to takeaway from this assignment was how to properly implement for-loops so that each loop calculates the x- and y-components of force that each celestial body exerts on one another. This has to be done before calculating the new velocities and positions. Other key concepts was implementing command line arguments so that in the terminal the program would accept one command line argument for a step timer and the total elapsed time. Lastly, we were required to use smart pointers to manage the lifetime of our celestial body objects and implement a step function within our universe file.

5.3 Learnings

PS2b had a lot going on for an assignment and I learned a lot going in blind and learning from outside resources such as other schools that had the exact assignment. I learned how to internally represent the forces exerted and manipulate them using accessors and mutators from within the universe file. Certain forces had to be calculated first such as the forces exerted on one another celestial body from Newton's Third Law of Gravitation before we could calculate the velocity, acceleration, and finally the position. Within each step, I learned that once we had calculated the next position of each planet, we could then incorporate it within SFML to calculate each celestial object's position each consecutive frame. Once all of that was complete, the next step was how to output the final properties of each planet to the same text file.

5.4 Output

Figure 5: Image of the solar system in motion



5.5 PS2b Makefile

5.5.1 Makefile

```
1 CC = g++
2 CFLAGS = -pedantic --std=c++14
3 LIBS = -lsfml-graphics -lsfml-window -lsfml-system
4 DEPS = CelestialBody.h
5
6 #%.o: %.cpp $(DEPS)
7 #     $(CC) $(CFLAGS) -c $<
8
9 all: NBody
10
11 NBody: main.o CelestialBody.o
12     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
13
14 main.o: main.cpp Universe.h
15     $(CC) $(CFLAGS) -c main.cpp $(LIBS)
16
17 CelestialBody.o: CelestialBody.cpp CelestialBody.h
18     $(CC) $(CFLAGS) -c CelestialBody.cpp $(LIBS)
19
20 clean:
21     rm *.o NBody
```


5.6 PS2b Main Source

5.6.1 main.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps2a>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <2.12.2022>*/
6  /*Sources Of Help: <Daly's Comp4 Discord Server /
   Boost Documentation>*/
7  /*
   ****
   */
8
9  #include <iostream>
10 #include <fstream>
11 #include <string>
12 #include <math.h>
13
14 #include "CelestialBody.h"
15 #include "Universe.h"
16
17 int main (int argc, char* argv[]) {
18
19     /*Read in the first two lines of planets.txt (
       the number of celestial bodies then the
       universe radius*/
20     int num_body;
21     std::cin >> num_body;
22     std::cout << num_body << std::endl; // Testing
       if it read correctly
23
24     double universe_radius;
25     std::cin >> universe_radius;
26     std::cout << universe_radius<< std::endl;
27
28     /*Time variables*/
29     double elapsedTime = 0.0;
30     double simulationTime = atof(argv[2]);
```

```

31     double endTime = atof(argv[1]);
32     //double simulationTime = 25000.0;
33     //double endTime = 157788000.0;
34
35     /*General force variables*/
36     double radius;
37     double force;
38     const double G = 6.67e-11;
39
40     /*Instantiate the Universe*/
41     Universe solar_system(universe_radius);
42
43     /*Create RenderWindow*/
44     sf::RenderWindow window(sf::VideoMode(500, 500),
45                             "Solar System");
46     sf::Vector2u window_size = window.getSize(); //
47         Store render window x,y coordinates
48
49     /*Manually create CelestialBody objects*/
50     //CelestialBody sun(0, 0, 0, 0, 0, "sun.gif");
51     //CelestialBody earth(1.4960e+11, 0, 0, 0, 0, "
52         earth.gif");
53     //CelestialBody mars(2.2790e+11, 0, 0, 0, 0, "
54         mars.gif");
55     //CelestialBody mercury(5.7900e+10, 0, 0, 0, 0,
56         "mercury.gif");
57
58     /*Have the Universe class instantiate new
59         CelestialBody objects*/
60     //solar_system.body.push_back (new CelestialBody
61         (2.2790e+11, 0, 0, 0, 0, "mars.gif"));
62     //solar_system.body.push_back (new CelestialBody
63         (5.7900e+10, 0, 0, 0, 0, "mercury.gif"));
64     //solar_system.body.push_back (new CelestialBody
65         (1.4960e+11, 0, 0, 2.9800e+04, 5.9740e+24, "
66         earth.gif"));
67     //solar_system.body.push_back (new CelestialBody
68         (0.0000e+00, 0, 0, 0, 0, "sun.gif"));
69     //solar_system.body.push_back (new CelestialBody
70         (1.0820e+11, 0, 0, 0, 0, "venus.gif"));

```

```

59
60
61     /*For loop to instantiate new celestial bodies,
        initialize the parameters from the overloaded
62     input stream operator, then load the image,
        texture, and sprite*/
63
64
65     for(int i = 0; i < num_body; i++) {
66         solar_system.body.push_back (new
            CelestialBody);
67         std::cin >> solar_system.body[i];
68         solar_system.body[i]->loadSprite();
69     }
70
71     window.setFramerateLimit(60.0);
72
73     while (window.isOpen())
74     {
75         sf::Event event;
76         while (window.pollEvent(event))
77         {
78             if (event.type == sf::Event::Closed)
79                 window.close();
80         }
81
82         if (elapsedTime <= endTime) {
83
84             /*Calculate the net forces using nested
                loop. This will calculate
85             the force body 'j' will exert on body 'i
                ' except when i=j. Forces
86             have to be calculated before calculating
                the new velocities and
87             positions.*/
88             for (int i = 0; i < num_body; i++) {
89                 solar_system.body[i]->setFx(0.0);
90                 solar_system.body[i]->setFy(0.0);
91                 for (int j = 0; j < num_body; j++) {
92                     if (i != j) {

```

```

93
94     radius = sqrt(pow((
        solar_system.body[j]->
        getXpos() - solar_system.
        body[i]->getXpos()), 2)
95         + pow((
            solar_system.
            body[j]->
            getYpos() -
            solar_system.
            body[i]->
            getYpos()),
            2));

96
97     force = (G * solar_system.
        body[i]->getMass() *
        solar_system.body[j]->
        getMass())/pow(radius, 2)
        ;

98
99     solar_system.body[i]->setFx(
        solar_system.body[i]->
        getFx() + force * ((
        solar_system.body[j]->
        getXpos() - solar_system.
        body[i]->getXpos())/
        radius));
100    solar_system.body[i]->setFy(
        solar_system.body[i]->
        getFy() + force * ((
        solar_system.body[j]->
        getYpos() - solar_system.
        body[i]->getYpos())/
        radius));

101
102    }
103 }
104 }
105
106 /*Loop to calculate new velocity and

```

```

        position with the universe's step
        function*/
107     for(int i = 0; i < num_body; i++) {
108         solar_system.step(simulationTime ,
            solar_system.body[i]);
109     }
110
111     elapsedTime += simulationTime;
112
113 }
114
115 /*Loop to plot the bodies after calculating
    the new position*/
116 for(int i = 0; i < num_body; i++) {
117     window.draw(*solar_system.body[i]);
118     solar_system.body[i]->renderToWindow(
        window_size , universe_radius);
119 }
120 std::cout << elapsedTime << endl;
121 window.display();
122 window.clear(sf::Color::Black);
123
124 }
125
126 fstream log;
127 log.open("planets.txt", fstream::app);
128 log << endl << "Simulation after " << endTime/
    simulationTime << " steps." << endl;
129 log << num_body << endl;
130 log << universe_radius << endl;
131
132 for(int i = 0; i < num_body; i++) {
133     log <<     solar_system.body[i]->getXpos() <<
        "      "
134         <<     solar_system.body[i]->getYpos() <<
        "      "
135         <<     solar_system.body[i]->getXvel() <<
        "      "
136         <<     solar_system.body[i]->getYvel() <<
        "      "

```

```
137         <<    solar_system.body[i]->getMass() <<
           "      "
138         <<    solar_system.body[i]->getFileName()
           << endl;
139
140     }
141
142     log.close();
143
144     return 0;
145 }
```

5.7 PS2b CelestialBody Source

5.7.1 CelestialBody.cpp

```
1 #include <iostream>
2 #include <string>
3
4 #include <SFML/System.hpp>
5 #include <SFML/Window.hpp>
6 #include <SFML/Graphics.hpp>
7
8 #include "CelestialBody.h"
9
10 CelestialBody::CelestialBody() { }
11
12 CelestialBody::CelestialBody(double xp, double yp,
13     double xv, double yv, double m, std::string img)
14 {
15     xpos = xp;
16     ypos = yp;
17     xvel = xv;
18     yvel = yv;
19     mass = m;
20     filename = img;
21 }
22
23
24 void CelestialBody::renderToWindow(const sf::
25     Vector2u& window_size, double universe_radius) {
26     /*Translate viewport to Cartesian coordinate
27     system*/
28     double xNew = (((universe_radius*2)/2) + xpos) *
29         (window_size.x/(universe_radius*2));
30     double yNew = (((universe_radius*2)/2) - ypos) *
31         (window_size.y/(universe_radius*2));
32
33     /*Gets bounding box of sprite and sets origin to
34     center by dividing width/height by 2*/
```

```

31     sf::FloatRect sprite_dimensions = sprite.
        getGlobalBounds();
32     sprite.setOrigin(sprite_dimensions.width/2,
        sprite_dimensions.height/2);
33
34     sprite.setPosition(xNew, yNew);
35
36 }
37
38 void CelestialBody::loadSprite() {
39
40     sf::Image image;
41
42     if (!image.loadFromFile(filename))
43         std::cout << "Cannot load image file." <<
            std::endl;
44
45     texture.loadFromImage(image);
46
47     sprite.setTexture(texture);
48
49 }
50
51 /*Mutator Functions*/
52 void CelestialBody::setXpos(double x) {
53
54     xpos = x;
55
56 }
57
58 double CelestialBody::getXpos() {
59
60     return xpos;
61
62 }
63
64 void CelestialBody::setYpos(double y) {
65
66     ypos = y;
67

```



```

68 }
69
70 double CelestialBody::getYpos() {
71
72     return ypos;
73 }
74 }
75
76 void CelestialBody::setXvel(double xv) {
77
78     xvel = xv;
79 }
80 }
81
82 double CelestialBody::getXvel() {
83
84     return xvel;
85 }
86 }
87
88
89 void CelestialBody::setYvel(double yv) {
90
91     yvel = yv;
92 }
93 }
94
95 double CelestialBody::getYvel() {
96
97     return yvel;
98 }
99 }
100
101
102 void CelestialBody::setMass(double m) {
103
104     mass = m;
105 }
106 }
107

```

```

108 double CelestialBody::getMass() {
109
110     return mass;
111 }
112 }
113
114 void CelestialBody::setFx(double fx) {
115     Fx = fx;
116 }
117
118 double CelestialBody::getFx() {
119     return Fx;
120 }
121
122 void CelestialBody::setFy(double fy) {
123     Fy = fy;
124 }
125
126 double CelestialBody::getFy() {
127     return Fy;
128 }
129
130 void CelestialBody::setFileName(std::string file) {
131
132     filename = file;
133 }
134 }
135
136 std::string CelestialBody::getFileName() {
137     return filename;
138 }

```

5.8 PS2b CelestialBody Header

5.8.1 CelestialBody.h

```
1 #ifndef CelestialBody_H
2 #define CelestialBody_H
3 #include <iostream>
4 #include <string>
5
6 #include <SFML/System.hpp>
7 #include <SFML/Window.hpp>
8 #include <SFML/Graphics.hpp>
9
10
11 class CelestialBody : public sf::Drawable
12 {
13     public:
14         /*Constructors*/
15         CelestialBody();
16         CelestialBody(double xp, double yp, double
            xv, double yv, double m, std::string img)
            ;
17
18         /*Mutator Functions*/
19         void setXpos(double x);
20         void setYpos(double y);
21         void setXvel(double xv);
22         void setYvel(double yv);
23         void setMass(double m);
24         void setFileName(std::string file);
25
26         void setFx(double fx);
27         void setFy(double fy);
28
29         /*Accessor Functions*/
30         double getXpos();
31         double getYpos();
32         double getXvel();
33         double getYvel();
34         double getMass();
```

```

35         std::string getFileName();
36
37         double getFx();
38         double getFy();
39
40
41         /*Load the image file and set the texture
           and sprite*/
42         void loadSprite();
43
44         /*Translate viewport to Cartesian
           coordinates and set sprite's origin (0,0)
           to be centered*/
45         void renderToWindow(const sf::Vector2u&
           window_size, double universe_radius);
46
47     private:
48         /*Private virtual void draw function*/
49         virtual void draw(sf::RenderTarget& target,
           sf::RenderStates states) const
50     {
51         target.draw(sprite);
52     }
53
54         /*Private member variables*/
55         double xpos;
56         double ypos;
57         double xvel;
58         double yvel;
59         double mass;
60
61         double Fy;
62         double Fx;
63
64         std::string filename;
65
66         /*Sprite and texture member variables*/
67         sf::Sprite sprite;
68         sf::Texture texture;
69 };

```

```
70  
71 #endif
```

5.9 PS2b Universe Header

5.9.1 Universe.h

```
1 #ifndef Universe_H
2 #define Universe_H
3 #include <iostream>
4 #include <string>
5
6 #include <SFML/System.hpp>
7 #include <SFML/Window.hpp>
8 #include <SFML/Graphics.hpp>
9
10 using namespace std;
11
12
13 class Universe : public sf::Drawable
14 {
15     public:
16         /*Constructor(s)*/
17         Universe(double rad);
18
19         /*Vector of pointers to objects (
20             CelestialBody)*/
21         std::vector<CelestialBody*> body;
22
23         /*Overloaded input stream operator*/
24         friend istream& operator>>(std::istream&
25             input, CelestialBody *&body);
26
27         /*Step function to simulate physics of
28             CelestialBodies*/
29         void step(double simulationTime,
30             CelestialBody *&body);
31
32     private:
33         virtual void draw(sf::RenderTarget& target,
34             sf::RenderStates states) const
35     {
```

```

32         target.draw(sprite);
33     }
34     double win_size;
35
36     double radius;
37     sf::Sprite sprite;
38     sf::Texture texture;
39
40     const double grav_constant = 6.67e-11;
41 };
42
43
44 Universe::Universe(double rad) {
45     radius = rad;
46 }
47
48 /*Overloaded istream operator to take inputs. Passes
49    the pointer(*) to object by reference(&) to
50    avoid copying*/
51 istream& operator>>(istream& input, CelestialBody *&
52    body) {
53
54     double _xpos;
55     double _ypos;
56     double _xvel;
57     double _yvel;
58     double _mass;
59     std::string _filename;
60
61     input >> _xpos >> _ypos >> _xvel >> _yvel >>
62         _mass >> _filename;
63
64     body->setXpos(_xpos);
65     body->setYpos(_ypos);
66     body->setXvel(_xvel);
67     body->setYvel(_yvel);
68     body->setMass(_mass);
69     body->setFileName(_filename);
70
71     return input;

```

```

68 }
69
70 void Universe::step(double dT, CelestialBody *&body)
71 {
72     // a = F/m
73     double xA = body->getFx()/body->getMass();
74     double yA = body->getFy()/body->getMass();
75
76     // v = v0 + dT*a
77     body->setXvel(body->getXvel() + dT*xA);
78     body->setYvel(body->getYvel() + dT*yA);
79
80     // s = s0 + dT*v
81     body->setXpos(body->getXpos() + dT*body->getXvel
82                 ());
83     body->setYpos(body->getYpos() + dT*body->getYvel
84                 ());
85 }
86
87 /*Passing a pointer to the object (NOT USED)*/
88 void Universe::renderToWindow(CelestialBody *&body
89 )
90 {
91     body->sprite.setPosition(200, 200);
92 }
93 }
94 */
95
96 #endif

```


6 PS3: Triangle Fractal

6.1 Overview

PS3 had us using the SFML library again but this time it was also a refresher on using recursion. We would be created a variation of inverted version of the Sierpensi's triangle using SFML. We are tasked with creating a recursive function called 'fTree()' and a main function that calls that particular function to create a series of triangles to represent the Sierpensi triangle. The program would take two command-line arguments, one for the length of the side of the base triangle, and one for the depth of recursion.

6.2 Key Concepts


PS3's key concepts included implementing recursion with SFML to incorporate recursive graphics within the display window. There were multiple ways of implementing this, such as the recursion method to draw the triangles and how different methods of positioning each triangle to represent the Sierpensi's triangle.

6.3 Learnings

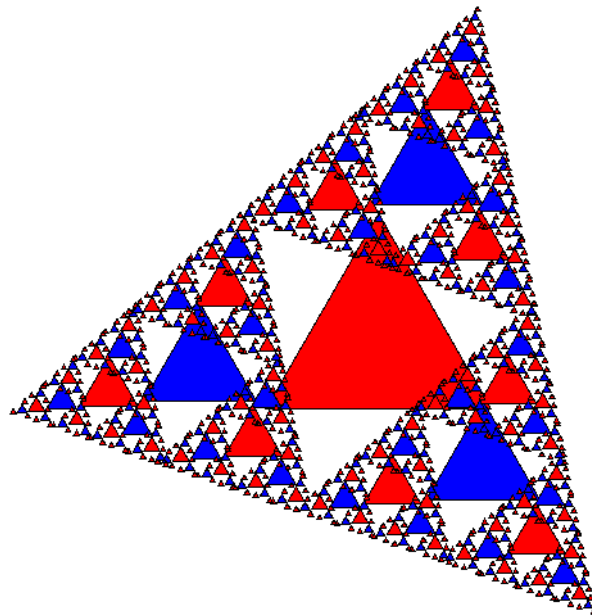
For this assignment, it served as a quick crash course on recursion for those who had forgotten but implemented it in a way that it worked with SFML to display recursive graphics. My approach to this assignment using the CircleShape class provided by SFML to draw a circle with three sides, also known as a triangle. I would then position triangle in the center, and proceed to get the global bounds of the base triangle to later pass the coordinates to the recursive function to properly draw the next three triangles in their proper position to represent the Sierpensi triangle. Coming out of this assignment, I learned how to incorporate recursion with SFML to create intricate looking graphics. My code for this assignment could have been better, since the position of the triangles were a bit off each call to the recursive function, but it did make for some interesting final results.

6.4 Output

Figure 6: Sierpinski's triangle with length 100 and recursion depth of 7

 Sierpinski's Triangle

— □ ×



6.5 PS3 Makefile

6.5.1 Makefile

```
1 CC = g++
2 CFLAGS = -pedantic --std=c++14
3 LIBS = -lsfml-graphics -lsfml-window -lsfml-system
4 DEPS = Triangle.h
5
6 #%.o: %.cpp $(DEPS)
7 #     $(CC) $(CFLAGS) -c $<
8
9 all: TFractal
10
11 TFractal: TFractal.o Triangle.o
12     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
13
14 TFractal.o: TFractal.cpp Triangle.h
15     $(CC) $(CFLAGS) -c TFractal.cpp $(LIBS)
16
17 Triangle.o: Triangle.cpp Triangle.h
18     $(CC) $(CFLAGS) -c Triangle.cpp $(LIBS)
19
20 lint:
21     cpplint --quiet *.cpp *.h
22
23 clean:
24     rm *.o TFractal
```

6.6 PS3 TFractal Source

6.6.1 TFractal.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps3>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <2.28.2022>*/
6  /*Sources Of Help: <StackOverflow>*/
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9
10 #include <iostream>
11 #include <math.h>
12
13 #include "Triangle.h"
14
15 void fTree(sf::RenderTarget &target, double radius,
    int depth, Triangle &frac);
16
17 int main(int argc, char* argv[]) {
18     double length = std::stod(argv[1]);
19     int depth = std::stoi(argv[2]);
20
21     sf::RenderWindow window(sf::VideoMode(750, 750), "
        Sierpinski's Triangle");
22     // Store render window x,y coordinates
23     sf::Vector2u window_size = window.getSize();
24
25     Triangle fractal;
26
27     while (window.isOpen()) {
28         sf::Event event;
29         while (window.pollEvent(event)) {
30             if (event.type == sf::Event::Closed)
31                 window.close();
32         }
33         fTree(window, length, depth, fractal);
```

```

34     window.display();
35     window.clear(sf::Color::White);
36 }
37     return 0;
38 }
39
40 void fTree(sf::RenderTarget &target, double radius,
    int depth, Triangle &frac) {
41     sf::CircleShape circle(radius, 3);
42     circle.setOutlineColor(sf::Color::Black);
43     circle.setOutlineThickness(1);
44
45     sf::FloatRect circ_bounds = circle.
        getGlobalBounds();
46     circle.setOrigin(circ_bounds.width/2,
        circ_bounds.height/2);
47     circle.setPosition(target.getSize().x / 2.f,
        target.getSize().y / 2.f);
48
49     frac.drawfTree(target, depth, circle);
50 }

```

6.7 PS3 Triangle Source

6.7.1 Triangle.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps3>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <2.28.2022>*/
6  /*Sources Of Help: <StackOverflow>*/
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9  #include <iostream>
10 #include <string>
11
12 #include <SFML/System.hpp>
13 #include <SFML/Window.hpp>
14 #include <SFML/Graphics.hpp>
15
16 #include "Triangle.h"
17
18 Triangle::Triangle() {}
19
20
21 void Triangle::drawTree(sf::RenderTarget& target,
22                        int depth,
23                        sf::CircleShape& base) {
24     if (depth < 1) return;
25
26     if (depth % 2 == 0) {
27         base.setFillColor(sf::Color::Blue);
28     } else {
29         base.setFillColor(sf::Color::Red);
30     }
31     target.draw(base);
32
33     auto rad = base.getRadius();
34     sf::FloatRect tBounds = base.
        getGlobalBounds();
```

```

35
36     auto leftT = base;
37     leftT.setRadius(rad/2);
38     leftT.setOrigin(leftT.getGlobalBounds().
        width/2,
39                     leftT.getGlobalBounds().
        width/2);
40     leftT.move(tBounds.width/2, tBounds.
        height*.75);
41     drawfTree(target, depth-1, leftT);
42
43     auto rightT = base;
44     rightT.setRadius(rad/2);
45     rightT.setOrigin(rightT.getGlobalBounds
        ().width/2,
46                     rightT.getGlobalBounds
        ().width/2);
47     rightT.move(-(tBounds.width*.75), (
        tBounds.height*.25));
48     drawfTree(target, depth-1, rightT);
49
50
51     auto bottomT = base;
52     bottomT.setRadius(rad/2);
53     bottomT.setOrigin(bottomT.
        getGlobalBounds().width/2,
54                     bottomT.
        getGlobalBounds().
        width/2);
55     bottomT.move(tBounds.width*.25, -(
        tBounds.height*.75));
56     drawfTree(target, depth-1, bottomT);
57 }

```

6.8 PS3 Triangle Header

6.8.1 Triangle.h

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps3>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <2.28.2022>*/
6  /*Sources Of Help: <StackOverflow>*/
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9  #ifndef _HOME_JHUA_COMPUTING_IV_PS3_TRIANGLE_H_
10 #define _HOME_JHUA_COMPUTING_IV_PS3_TRIANGLE_H_
11 #include <iostream>
12 #include <string>
13
14 #include <SFML/System.hpp>
15 #include <SFML/Window.hpp>
16 #include <SFML/Graphics.hpp>
17
18 class Triangle : public sf::Drawable {
19 public:
20     Triangle();
21
22     void drawTree(sf::RenderTarget& target, int
                depth,
23                 sf::CircleShape& base);
24
25 private:
26     /*Private virtual void draw function*/
27     virtual void draw(sf::RenderTarget& target,
28                       sf::RenderStates states)
29                     const {}
30
31     double length;
32     int depth;
33 };
```



```
34  
35 #endif // _HOME_JHUA_COMPUTING_IV_PS3_TRIANGLE_H_
```

7 PS4a: CircularBuffer

7.1 Overview

PS4a is part of a two-part assignment where the first part we were tasked with creating a circular buffer. This circular buffer would later serve as the data structure to contain samples to simulate the Karplus-Strong algorithm to simulate the plucking of a guitar. In short, the circular buffer or ring buffer is a queue where both ends are connected to simulate a 'ring' so to speak. How we go about implementing this is left to our own interpretation as there are several ways to implement it as well using the Boost test framework to test the methods within it.

7.2 Key Concepts

The main concepts to take away from this assignment was implementing the proper data structure so that the ring or circular buffer would function correctly. Lots of tests were done such as ensuring that the circular buffer would properly dequeue and enqueue data and would loop back to the start of the buffer when it was full. We revisit the Boost framework again to perform all these tests, in addition to checking if the buffer was full, empty, and the peek method. All of these were essential for the next part of the assignment to simulate sound samples of the plucking of a guitar.

7.3 Learnings

As stated earlier, the precedent set for this class was that a lot of the assignments were left up to our own interpretation on how to implement an algorithm. The standard template library from C++ provides a lot of nice things to use and basis for the assignment. I chose to implement it as instantiating as an array where the start and end of the array were connected, and had pointers or trackers named as front and rear to keep track of the ring buffer. Throughout the assignment, I learned or relearned more specifically a deeper understanding of data structures and its inner workings by making a circular buffer; something that was never mentioned or covered in previous courses.

7.4 Output

Figure 7: Running output with a main driver and Boost passing test cases

```
jhua@DESKTOP-75DLUJV:~/computing_IV/ps4a$ ./main
Item dequeued was: 1
Item dequeued was: 2
Ring buffer full.
jhua@DESKTOP-75DLUJV:~/computing_IV/ps4a$ ./ps4a
Running 2 test cases...

*** No errors detected
jhua@DESKTOP-75DLUJV:~/computing_IV/ps4a$
```

7.5 PS4a Makefile

7.5.1 Makefile

```
1 CC = g++
2 CFLAGS = -pedantic --std=c++17
3 LIBS = -lboost_unit_test_framework
4 DEPS = CircularBuffer.h
5
6 #%.o: %.cpp $(DEPS)
7 #     $(CC) $(CFLAGS) -c $<
8
9 all: main ps4a
10
11 ps4a: test.o CircularBuffer.o
12     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
13
14 main: main.o CircularBuffer.o
15     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
16
17 main.o: main.cpp CircularBuffer.h
18     $(CC) $(CFLAGS) -c main.cpp $(LIBS)
19
20 CircularBuffer.o: CircularBuffer.h
21     $(CC) $(CFLAGS) -c CircularBuffer.cpp $(LIBS)
22
23 test.o: CircularBuffer.h
24     $(CC) $(CFLAGS) -c test.cpp $(LIBS)
25
26 clean:
27     rm *.o main ps4a
```

7.6 PS4a Main Source

7.6.1 main.cpp

```
1 #include <iostream>
2 #include <string>
3 #include <stdint.h>
4
5 #include "CircularBuffer.h"
6
7 int main() {
8
9     CircularBuffer ring_buffer(5);
10
11     //std::cout << "hello world" << std::endl;
12     ring_buffer.enqueue(1);
13     ring_buffer.enqueue(2);
14     ring_buffer.enqueue(3);
15     ring_buffer.enqueue(4);
16     ring_buffer.enqueue(5);
17     std::cout << "Item dequeued was: " <<
18         ring_buffer.dequeue() << std::endl;
19     std::cout << "Item dequeued was: " <<
20         ring_buffer.dequeue() << std::endl;
21     ring_buffer.enqueue(7);
22     ring_buffer.enqueue(8);
23
24     //std::cout << ring_buffer.size() << std::endl;
25     //std::cout << ring_buffer.peek_front() << std::
26         endl;
27     //std::cout << ring_buffer.peek_rear() << std::
28         endl;
29
30     if(ring_buffer.isFull()) {
31         std::cout << "Ring buffer full." << std::
32             endl;
33     } else {
34         std::cout << "Ring buffer has space." << std
35             ::endl;
36     }
37 }
```

```
31     return 0;  
32 }
```

7.7 PS4a CircularBuffer Source

7.7.1 CircularBuffer.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps4a>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <3.20.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9
10 #include "CircularBuffer.h"
11
12 CircularBuffer::CircularBuffer(size_t capacity) {
13     if (capacity < 1) {
14         throw std::invalid_argument
15             ("Buffer capacity must be greater than zero.
16             ");
17     } else {
18         buffer = new int16_t[capacity];
19         buffer_size = capacity;
20     }
21 }
22 CircularBuffer::~CircularBuffer() {
23     delete buffer;
24 }
25
26 size_t CircularBuffer::size() {
27     return buffer_size;
28 }
29
30 bool CircularBuffer::isFull() {
31     return(front == (rear + 1) % buffer_size);
32 }
33
34 bool CircularBuffer::isEmpty() {
```

```

35     return(front == rear);
36 }
37
38 void CircularBuffer::enqueue(int16_t x) {
39     if (isFull()) {
40         throw std::runtime_error
41             ("Cannot enqueue to a ring buffer that is
42              full.");
43     } else if (isEmpty()) {
44         front = 0;
45         rear = (rear + 1) % buffer_size;
46         buffer[rear] = x;
47     } else {
48         rear = (rear + 1) % buffer_size;
49         buffer[rear] = x;
50     }
51 }
52 int16_t CircularBuffer::dequeue() {
53     int16_t buf_value;
54     if (isEmpty()) {
55         throw std::runtime_error
56             ("Dequeue: nothing to dequeue as the ring
57              buffer is empty.");
58     } else if (front == rear) { // Only one element
59         // in the ring buffer
60         front = rear = -1;
61     } else {
62         buf_value = buffer[front];
63         buffer[front] = 0;
64         front = (front + 1) % buffer_size;
65     }
66     return buf_value;
67 }
68 int16_t CircularBuffer::peek() {
69     if (isEmpty()) {
70         throw std::runtime_error
71             ("Peek: nothing to peek as ring buffer is
72              empty.");

```



```
71     } else {  
72         return buffer[front];  
73     }  
74 }  
75  
76 int16_t CircularBuffer::peek_front() {  
77     return buffer[front];  
78 }  
79  
80 int16_t CircularBuffer::peek_rear() {  
81     return buffer[rear];  
82 }
```

7.8 PS4a CircularBuffer Header

7.8.1 CircularBuffer.h

```
1 /*Name: <John Hua>*/
2 /*Course name: <COMP.2040>*/
3 /*Assignment: <ps4a>*/
4 /*Instructor's name: <Dr. James Daly>*/
5 /*Date: <3.19.2022>*/
6 /*Sources Of Help: */
7 /*Copyright 2022 <John Hua>
8 /*****
   */
9 #ifndef
   _HOME_JHUA_COMPUTING_IV_PS4A_CIRCULARBUFFER_H_
10 #define
   _HOME_JHUA_COMPUTING_IV_PS4A_CIRCULARBUFFER_H_
11 #include <stdint.h>
12 #include <iostream>
13 #include <string>
14
15 class CircularBuffer {
16 public:
17     explicit CircularBuffer(size_t capacity);
18     ~CircularBuffer();
19     size_t size();
20     bool isEmpty();
21     bool isFull();
22     void enqueue(int16_t x);
23
24     int16_t dequeue();
25     int16_t peek();
26     int16_t peek_front();
27     int16_t peek_rear();
28
29 private:
30     int16_t* buffer;
31     size_t front = -1;
32     size_t rear = -1;
33     size_t buffer_size;
```

```
34 };  
35  
36 #endif //  
    _HOME_JHUA_COMPUTING_IV_PS4A_CIRCULARBUFFER_H_
```

7.9 PS4a Boost Source

7.9.1 test.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps4a>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <3.19.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9
10 #include <iostream>
11 #include <string>
12
13 #include "CircularBuffer.h"
14
15 #define BOOST_TEST_DYN_LINK
16 #define BOOST_TEST_MODULE Main
17 #include <boost/test/unit_test.hpp>
18
19 /*Check that invalid argument exception is thrown
    when attempting to
20 initialize ring buffer with size < 0*/
21 BOOST_AUTO_TEST_CASE(checkExcpt_bufferSize) {
22     BOOST_REQUIRE_THROW(CircularBuffer buffer_zero(0),
        std::invalid_argument);
23     BOOST_REQUIRE_NO_THROW(CircularBuffer buffer_zero
        (1));
24 }
25
26 /*Check enqueue, dequeue, and peek functions on an
    empty and full buffer
27 of size 5.*/
28 BOOST_AUTO_TEST_CASE(checkExcpt_queue) {
29     CircularBuffer fullBuffer(5);
30     CircularBuffer emptyBuffer(5);
31     for (auto i = 1; i <= fullBuffer.size(); i++) {
```

```
32     fullBuffer.enqueue(i);
33 }
34 BOOST_REQUIRE_THROW(fullBuffer.enqueue(1), std::
    runtime_error);
35 BOOST_REQUIRE_THROW(emptyBuffer.dequeue(), std::
    runtime_error);
36 BOOST_REQUIRE_THROW(emptyBuffer.peek(), std::
    runtime_error);
37 }
```

8 PS4b: String Sound

8.1 Overview

PS4b is the second part of the two-part assignment where we now have to use the circular buffer we created back in part A to implement the Karplus-Strong guitar string simulation. This would incorporate the SFML audio library to generate and play a stream of string samples back to us using keyboard inputs. A new class is introduced in this assignment called StringSound with new constructors and methods to calculate the string of samples given by a frequency. Similar to the N-Body simulation assignment, a method called "tic()" is used to advance the simulation one step and a method called "pluck()" to pluck the guitar string by replacing the buffer with arbitrary values.

8.2 Key Concepts

Key concepts in PS4b included incorporating the circular buffer we made in the first part A with SFML's audio library and using PulseAudio. We would create a program called KSGuitarSim that supports a total of 37 notes on the chromatic scale from 110Hz to 880Hz.

8.3 Learnings

For this assignment, I could not completely finish it since I was having trouble setting up PulseAudio. Therefore, I could not fully test the audio functionality of the assignment but instead could only create a main driver function that drove around the circular buffer and tested the methods in the StringSound class such as the pluck and tic methods. As for pluck method where we were required to enqueue the buffer with random values, I had used a randomizer method that filled the buffer with random values from most negative `int16_t` value to the most positive, -32768 to 32767, in a uniform distribution. This method which I had learned from Stack Exchange would more than likely be useful for other projects or assignments.

8.4 Output

Figure 8: Main driver that tests the functionality of StringSound/Circular-Buffer

```
jhua@DESKTOP-75DLUJV:~/computing_IV/ps4b$ ./main
Item dequeued was: 1
Item dequeued was: 2
Ring buffer full.
5
0
StringSound buffer size initialized to: 882
Testing pluck method that it enqueues random values correctly:
-7785
-9449
31395
-29894
29679
9184
25780
-21592
3624
-10391
Current sample: -7785
Sample after call to tic(): -9449
Current time: 1
```

8.5 PS4b Makefile

8.5.1 Makefile

```
1 CC = g++
2 CFLAGS = -pedantic --std=c++17
3 LIBS = -lboost_unit_test_framework -lsfml-graphics -
        lsFML-audio -lsfml-system -lsfml-window -lsfml-
        system
4 DEPS = CircularBuffer.h StringSound.h
5
6 %.o: %.cpp $(DEPS)
7 # $(CC) $(CFLAGS) -c $<
8
9 all: ps4a main
10
11 ps4a: test.o CircularBuffer.o
12     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
13
14 main: main.o CircularBuffer.o StringSound.o
15     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
16
17 main.o: main.cpp CircularBuffer.h
18     $(CC) $(CFLAGS) -c main.cpp $(LIBS)
19
20 CircularBuffer.o: CircularBuffer.h
21     $(CC) $(CFLAGS) -c CircularBuffer.cpp $(LIBS)
22
23 StringSound.o: StringSound.h CircularBuffer.h
24     $(CC) $(CFLAGS) -c StringSound.cpp $(LIBS)
25
26 test.o: CircularBuffer.h
27     $(CC) $(CFLAGS) -c test.cpp $(LIBS)
28
29 #SSLite: SSLite.o CircularBuffer.o StringSound.o
30 # $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
31
32 #SSLite.o: CircularBuffer.h StringSound.h
33 # $(CC) $(CFLAGS) -c SSLite.cpp $(LIBS)
34
```



```
35 clean:
36     rm *.o main ps4a
```

8.6 PS4b Main Source

8.6.1 main.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps4b>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <3.28.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9
10 #include <stdint.h>
11 #include <iostream>
12 #include <string>
13 #include <random>
14
15 #include "CircularBuffer.h"
16 #include "StringSound.h"
17
18
19 int main() {
20     CircularBuffer ring_buffer(5);
21     CircularBuffer ring_buffer2(5);
22
23     ring_buffer.enqueue(1);
24     ring_buffer.enqueue(2);
25     ring_buffer.enqueue(3);
26     ring_buffer.enqueue(4);
27     ring_buffer.enqueue(5);
28     std::cout << "Item dequeued was: " <<
        ring_buffer.dequeue() << std::endl;
29     std::cout << "Item dequeued was: " <<
        ring_buffer.dequeue() << std::endl;
30     ring_buffer.enqueue(7);
31     ring_buffer.enqueue(8);
32     // std::cout << ring_buffer.size() << std::endl;
33     // std::cout << ring_buffer.peek_front() << std
```

```

        ::endl;
34 // std::cout << ring_buffer.peek_rear() << std::
    endl;
35
36 if (ring_buffer.isFull()) {
37     std::cout << "Ring buffer full." << std::
        endl;
38 } else {
39     std::cout << "Ring buffer has space." << std
        ::endl;
40 }
41
42 std::cout << ring_buffer2.size() << std::endl;
43 std::cout << ring_buffer2.buffer[2] << std::endl
    ;
44
45 StringSound string_test(50.0);
46 std::cout << "StringSound buffer size
    initialized to: "
47     << string_test._cb->size() << std::
        endl;
48 string_test.pluck();
49 std::cout << "Testing pluck method that it
    enqueues random values correctly: " << std::
        endl; //NOLINT
50 for (size_t i = 0; i < 10; i++) {
51     std::cout << string_test._cb->buffer[i] <<
        std::endl;
52 }
53
54 std::cout << "Current sample: " << string_test.
    sample() << std::endl;
55 string_test.tic();
56 std::cout << "Sample after call to tic(): " <<
    string_test.sample() << std::endl; //NOLINT
57 std::cout << "Current time: " << string_test.
    time() << std::endl;
58 return 0;
59 }

```

8.7 PS4b CircularBuffer Source

8.7.1 CircularBuffer.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps4b>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <3.20.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9
10 #include "CircularBuffer.h"
11 #define SAMPLES_PER_SEC 44100
12
13 CircularBuffer::CircularBuffer(size_t capacity) {
14     if (capacity < 1) {
15         throw std::invalid_argument
16             ("Buffer capacity must be greater than zero.
17             ");
18     } else {
19         buffer = new int16_t[capacity];
20         buffer_size = capacity;
21     }
22 }
23
24 CircularBuffer::~CircularBuffer() {
25     delete buffer;
26 }
27
28 size_t CircularBuffer::size() {
29     return buffer_size;
30 }
31
32 bool CircularBuffer::isFull() {
33     return(front == (rear + 1) % buffer_size);
34 }
```

```

35 bool CircularBuffer::isEmpty() {
36     return(front == rear);
37 }
38
39 void CircularBuffer::enqueue(int16_t x) {
40     if (isFull()) {
41         throw std::runtime_error
42             ("Cannot enqueue to a ring buffer that is
43              full.");
44     } else if (isEmpty()) {
45         front = 0;
46         rear = (rear + 1) % buffer_size;
47         buffer[rear] = x;
48     } else {
49         rear = (rear + 1) % buffer_size;
50         buffer[rear] = x;
51     }
52 }
53
54 int16_t CircularBuffer::dequeue() {
55     int16_t buf_value;
56     if (isEmpty()) {
57         throw std::runtime_error
58             ("Dequeue: nothing to dequeue as the ring
59              buffer is empty.");
60     } else if (front == rear) { // Only one element
61         // in the ring buffer
62         front = rear = -1;
63     } else {
64         buf_value = buffer[front];
65         buffer[front] = 0;
66         front = (front + 1) % buffer_size;
67     }
68     return buf_value;
69 }
70
71 int16_t CircularBuffer::peek() {
72     if (isEmpty()) {
73         throw std::runtime_error
74             ("Peek: nothing to peek as ring buffer is

```

```

        empty.");
72     } else {
73         return buffer[front];
74     }
75 }
76
77 int16_t CircularBuffer::peek_front() {
78     return buffer[front];
79 }
80
81 int16_t CircularBuffer::peek_rear() {
82     return buffer[rear];
83 }
84
85 void CircularBuffer::empty() {
86     front = -1;
87     rear = -1;
88 }

```

8.8 PS4b CircularBuffer Header

8.8.1 CircularBuffer.h

```
1 /*Name: <John Hua>*/
2 /*Course name: <COMP.2040>*/
3 /*Assignment: <ps4b>*/
4 /*Instructor's name: <Dr. James Daly>*/
5 /*Date: <3.19.2022>*/
6 /*Sources Of Help: */
7 /*Copyright 2022 <John Hua>
8 /*****
   */
9 #ifndef
   _HOME_JHUA_COMPUTING_IV_PS4B_CIRCULARBUFFER_H_
10 #define
   _HOME_JHUA_COMPUTING_IV_PS4B_CIRCULARBUFFER_H_
11 #include <stdint.h>
12 #include <iostream>
13 #include <string>
14
15 class CircularBuffer {
16 public:
17     explicit CircularBuffer(size_t capacity);
18     ~CircularBuffer();
19     size_t size();
20     bool isEmpty();
21     bool isFull();
22     void enqueue(int16_t x);
23     void empty();
24
25     int16_t dequeue();
26     int16_t peek();
27     int16_t peek_front();
28     int16_t peek_rear();
29
30     int16_t* buffer;
31     size_t front = -1;
32     size_t rear = -1;
33     size_t buffer_size;
```

```
34
35 private:
36 };
37
38
39 #endif //
    _HOME_JHUA_COMPUTING_IV_PS4B_CIRCULARBUFFER_H_
```


8.9 PS4b StringSound Source

8.9.1 StringSound.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps4b>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <3.28.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9  #include "StringSound.h"
10 #include <math.h>
11 #include <random>
12 #include <SFML/Graphics.hpp>
13 #include <SFML/System.hpp>
14 #include <SFML/Audio.hpp>
15 #include <SFML/Window.hpp>
16
17
18 #define SAMPLES_PER_SEC 44100
19
20
21 StringSound::StringSound(double frequency) {
22     _time = 0;
23     if (frequency < 1) {
24         throw std::invalid_argument
25             ("Frequency must be greater than 0 to
26              initialize circular buffer.");
27     } else {
28         _cb = new CircularBuffer(ceil(
29             SAMPLES_PER_SEC/frequency));
30     }
31 }
32
33 StringSound::StringSound(std::vector<sf::Int16> init
34     ) {
35     _time = 0;
```

```

33     if (init.size() < 1) {
34         throw std::invalid_argument
35             ("Passed vector's size must be greater than
36              zero.");
37     } else {
38         _cb = new CircularBuffer(init.size());
39         for (size_t i = 0; !_cb->isFull(); i++) {
40             _cb->enqueue(init[i]);
41         }
42     }
43
44 StringSound::~StringSound() {
45     delete _cb->buffer;
46 }
47
48 void StringSound::pluck() {
49     std::random_device rd;
50
51     if (_cb->isFull()) {
52         _cb->empty();
53     }
54     while (!_cb->isFull()) {
55         std::mt19937 rng(rd());
56         std::uniform_int_distribution<int16_t> uni
57             (-32768, 32767);
58         _cb->enqueue(uni(rng));
59     }
60
61 void StringSound::tic() {
62     auto n = 0.996 * 0.5 * (_cb->dequeue() + _cb->
63         peek());
64     _time++;
65 }
66 sf::Int16 StringSound::sample() {
67     return _cb->peek();
68 }
69

```

```
70 int StringSound::time() {  
71     return _time;  
72 }
```

8.10 PS4b StringSound Header

8.10.1 StringSound.h

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps4b>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <3.28.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9  #ifndef _HOME_JHUA_COMPUTING_IV_PS4B_STRINGSOUND_H_
10 #define _HOME_JHUA_COMPUTING_IV_PS4B_STRINGSOUND_H_
11
12 #include <stdint.h>
13 #include <iostream>
14 #include <string>
15 #include <vector>
16 #include <SFML/Graphics.hpp>
17 #include <SFML/System.hpp>
18 #include <SFML/Audio.hpp>
19 #include <SFML/Window.hpp>
20
21 #include "CircularBuffer.h"
22
23 class StringSound {
24 public:
25     explicit StringSound(double frequency);
26     explicit StringSound(std::vector<sf::Int16> init);
27     StringSound (const StringSound &obj) = delete;    //
        no copy const
28     ~StringSound();
29     void pluck();
30     void tic();
31     sf::Int16 sample();
32     int time();
33
34     CircularBuffer * _cb;
```

```
35  int _time;
36  private:
37 };
38
39 #endif //
    _HOME_JHUA_COMPUTING_IV_PS4B_STRINGSOUND_H_
```

8.11 PS4b Boost Source

8.11.1 test.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps4a>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <3.19.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9
10 #include <iostream>
11 #include <string>
12
13 #include "CircularBuffer.h"
14
15 #define BOOST_TEST_DYN_LINK
16 #define BOOST_TEST_MODULE Main
17 #include <boost/test/unit_test.hpp>
18
19 /*Check that invalid argument exception is thrown
    when attempting to
20 initialize ring buffer with size < 0*/
21 BOOST_AUTO_TEST_CASE(checkExcpt_bufferSize) {
22     BOOST_REQUIRE_THROW(CircularBuffer buffer_zero(0),
        std::invalid_argument);
23     BOOST_REQUIRE_NO_THROW(CircularBuffer buffer_zero
        (1));
24 }
25
26 /*Check enqueue, dequeue, and peek functions on an
    empty and full buffer
27 of size 5.*/
28 BOOST_AUTO_TEST_CASE(checkExcpt_queue) {
29     CircularBuffer fullBuffer(5);
30     CircularBuffer emptyBuffer(5);
31     for (auto i = 1; i <= fullBuffer.size(); i++) {
```

```
32     fullBuffer.enqueue(i);
33 }
34 BOOST_REQUIRE_THROW(fullBuffer.enqueue(1), std::
    runtime_error);
35 BOOST_REQUIRE_THROW(emptyBuffer.dequeue(), std::
    runtime_error);
36 BOOST_REQUIRE_THROW(emptyBuffer.peek(), std::
    runtime_error);
37 }
```

9 PS5: DNA Alignment

9.1 Overview

PS5 is an assignment that dealt with the alignment of a genetic sequence, a real-world application and fundamental problem in computational biology. Pair programming was optional for this assignment due to the sheer difficulty but I had opted to work alone since I did not have anybody to work with. Using dynamic programming, we were required to compute the optimal sequence between two genetic sequences and compute the edit distance or otherwise known as the alignment score. There were many approaches to this problem such as using recursion to arrive at the answer, but I had opted to use dynamic program with a $N \times M$ matrix using Needleman-Wunsch's method.

9.2 Key Concepts

The main key concepts was the real-world application of using dynamic programming to solve problems in other disciplines such as life sciences and biology. We were advised to use valgrind to as well to check for memory leaks and verify our algorithm's space usage. Other key concepts included utilizing an $N \times M$ matrix properly and accounting for all the cases to compute the optimal alignment between two genetic strings.

9.3 Learnings

The difficulty ceiling of this assignment was slightly higher than previous ones, so I had to go and do some research beforehand such as the concept of aligning two genetic sequences. This was important since it was the first step into visualizing the problem and then beginning to think on how to implement it in code. The concept of dynamic programming was new to me as well, and the assignment gave a deeper understanding on how to solve or approach a problem by dividing it into smaller subproblems which would later prove useful in other fields of work or challenges in the software engineering field.

9.4 Output

Figure 9: Main driver computes the optimal alignment and the edit distance of an arbitrary sequence

```
jhua@DESKTOP-75DLUJV:~/computing_IV/ps5$ ./EDistance < teststring.txt
Edit Distance = 7
TA-AGGT-CA
AACAGTTACC
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: 0.007986 seconds
Edit Distance = 7
```

9.5 PS5 Makefile

9.5.1 Makefile

```
1 CC = g++
2 CFLAGS = -pedantic --std=c++17
3 LIBS = -lsfml-graphics -lsfml-window -lsfml-system
4 DEPS = EDistance.h
5
6 #%.o: %.cpp $(DEPS)
7 #     $(CC) $(CFLAGS) -c $<
8
9 all: EDistance
10
11 EDistance: main.o EDistance.o
12     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
13
14 main.o: main.cpp EDistance.h
15     $(CC) $(CFLAGS) -c main.cpp $(LIBS)
16
17 EDistance.o: EDistance.cpp EDistance.h
18     $(CC) $(CFLAGS) -c EDistance.cpp $(LIBS)
19
20 clean:
21     rm *.o EDistance
```

9.6 PS5 Main Source

9.6.1 main.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps5>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <4.4.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9  #include <iostream>
10 #include <string>
11 #include <vector>
12 #include <SFML/System.hpp>
13
14 using namespace std;
15
16 #include "EDistance.h"
17
18 sf::Clock clock1;
19 sf::Time t;
20
21 int main() {
22
23     string gene_seq1;
24     cin >> gene_seq1;
25
26     string gene_seq2;
27     cin >> gene_seq2;
28
29     EDistance e1(gene_seq1, gene_seq2);
30     std::cout << "Edit Distance = " << e1.
        optDistance() << std::endl;
31     std::cout << e1.alignment() << std::endl;
32     e1.printTable();
33
34     t = clock1.getElapsedTime();
```

```

35     cout << "Execution time: " << t.asSeconds() << "
        seconds" << endl;
36     std::cout << "Edit Distance = " << e1.
        optDistance() << std::endl;
37     //cout << gene_seq1 << " " << gene_seq2 << endl;
38     return 0;
39 }

```

9.7 PS5 EDistance Source

9.7.1 EDistance.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps4a>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <3.19.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9  #include <iostream>
10 #include <string>
11 #include <vector>
12
13 #include "EDistance.h"
14
15 using namespace std;
16
17 #define PENALTY_GAP 2
18
19 string aseq_align = "";
20 string bseq_align = "";
21
22 EDistance::EDistance(string a, string b) :
23 optMatrix(a.length() + 1, vector<int>(b.length() +
24     1)) {
25     aseq = a;
26     bseq = b;
27     n = a.length();
28     m = b.length();
29
30     for (size_t i = 0; i <= m; i++)
31         optMatrix[0][i] = i * PENALTY_GAP;
32     for (size_t i = 0; i <= n; i++)
33         optMatrix[i][0] = i * PENALTY_GAP;
34 }
```

```

35 int EDistance::optDistance() {
36     for (size_t i = 1; i <= n; i++)
37     {
38         for (size_t j = 1; j <= m; j++)
39         {
40             optMatrix[i][j] = min(optMatrix[i-1][j-1] +
41                                   penalty(aseq[i-1], bseq[j-1]),
42                                   optMatrix[i-1][j]
43                                   + PENALTY_GAP,
44                                   optMatrix[i][j-1]
45                                   + PENALTY_GAP);
46         }
47     }
48     return optMatrix[n][m];
49 }
50
51 int EDistance::min(int a, int b, int c) {
52     if (a <= b && a <= c)
53         return a;
54     else if (b <= a && b <= c)
55         return b;
56     else
57         return c;
58 }
59
60 int EDistance::penalty(char a, char b) {
61     return a == b ? 0 : 1;
62 }
63
64 string EDistance::alignment() {
65     size_t i = aseq.length();
66     size_t j = bseq.length();
67     for (; i > 0 && j > 0; i--) {
68         if (optMatrix[i-1][j-1] + penalty(aseq[i -
69             1], bseq[j - 1]) == optMatrix[i][j]) {
70             aseq_align = aseq[i - 1] + aseq_align;
71             bseq_align = bseq[j - 1] + bseq_align;
72             // values.push_back(penalty(aseq_align[i
73                 ], bseq_align[j]));

```

```

70         j--;
71     } else if (optMatrix[i - 1][j] + PENALTY_GAP
72               == optMatrix[i][j]) {
73         aseq_align = aseq[i - 1] + aseq_align;
74         bseq_align = '-' + bseq_align;
75         //values.push_back(2);
76     } else {
77         aseq_align = '-' + aseq_align;
78         bseq_align = bseq[j - 1] + bseq_align;
79         //values.push_back(2);
80         j--;
81     }
82 }
83 while (i >= 1 && j == 0)
84 {
85     aseq_align = aseq[i - 1] + aseq_align;
86     bseq_align = '-' + bseq_align;
87     i--;
88 }
89 while (j >= 1 && i == 0)
90 {
91     aseq_align = '-' + aseq_align;
92     bseq_align = bseq[j - 1] + bseq_align;
93     j--;
94 }
95 return bseq_align + '\n' + aseq_align;
96 }
97 void EDistance::printTable() {
98     for(size_t i = 0; i < aseq_align.length(); i++)
99     {
100         if (aseq_align[i] == bseq_align[i]) {
101             values.push_back(0);
102         } else if (aseq_align[i] != bseq_align[i])
103         {
104             if (aseq_align[i] == '-' || bseq_align[i]
105                 == '-') {
106                 values.push_back(2);
107             } else {
108                 values.push_back(1);

```

```
106         }
107     }
108 }
109 for (size_t i = 0; i < aseq.length(); i++) {
110     cout << aseq_align[i] << " " << bseq_align[i]
111         << " " << values[i] << endl;
112 }
```


9.8 PS5 EDistance Header

9.8.1 EDistance.h

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps5>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <4.4.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9  #ifndef EDistance_H
10 #define EDistance_H
11 #include <iostream>
12 #include <string>
13 #include <vector>
14
15
16 class EDistance {
17 public:
18     EDistance(std::string a, std::string b);
19     int penalty(char a, char b);
20     int min(int a, int b, int c);
21     int optDistance();
22     std::string alignment();
23     void printTable();
24     friend std::istream& operator>>(std::istream&
        input, EDistance &gene) {
25         input >> gene.aseq >> gene.bseq;
26         return input;
27     }
28
29     std::string aseq;
30     std::string bseq;
31     int n;
32     int m;
33     std::vector<std::vector<int>> optMatrix;
34     std::vector<int> values;
```

```
35 private:
36 };
37
38 #endif //
    _HOME_JHUA_COMPUTING_IV_PS4B_CIRCULARBUFFER_H_
```

10 PS6: Random Writer

10.1 Overview

PS6 was an interesting assignment with a difficulty level about the same as the previous assignment where we were tasked with generating random or nonsense text given input text. This required using a map to store frequency counts for a 'kgram', a given word in the input text. This method simulated the Markov chain algorithm to determine the next word for each given kgram in the table, and would produce pseudo-random text that somewhat made sense depending on the Markov order.

10.2 Key Concepts

Key concepts of the random writer was utilizing a map to store frequency counts for a kgram to produce a probabilistic model of the input text, given a kgram series of letters. Other key concepts to take away from this assignment were the Markov chains and implementing it using a map storing the frequency counts of a kgram. We were given and required to implement two methods of `freq()` where one was to find the frequency of a kgram and the other was to find the frequency of a char following that particular given kgram. Following that, we also implemented a method called `kRand()` that would generate a random character following a given kgram. Lastly, we were also required to implement exception handling for any wrongly given inputs passed to these functions.

10.3 Learnings

One of the key things I learned was how to utilize a map from the C++ standard template library properly for this assignment. The assignment was also an introduction to Markov chains and Markov models, and the assignment in itself was interesting in how it incorporated Markov chains to produce pseudo-random text. Markov chains also have a lot of real-world applications such as pattern recognition, modelling random processes, and among other things. It was interesting to see and learn how the order of Markov could affect the pseudo-randomness and readability of the output text from the given input text.

10.4 Output

Figure 10: TextWriter producing pseudo-random text from a given input text file of Markov order 5

```
jhua@DESKTOP-75DLUJV:~/computing_IV/ps6$ ./test
Running 2 test cases...

*** No errors detected
jhua@DESKTOP-75DLUJV:~/computing_IV/ps6$ ./TextWriter 2 11 < hi.txt
There'llly o
jhua@DESKTOP-75DLUJV:~/computing_IV/ps6$ ./TextWriter 5 100 < hi.txt
THE two started light; he could deftly little away to ventually if I did I? And Joe. "I was far int
jhua@DESKTOP-75DLUJV:~/computing_IV/ps6$ █
```

10.5 PS6 Makefile

10.5.1 Makefile

```
1 CC = g++
2 CFLAGS = -pedantic --std=c++17
3 LIBS = -lboost_unit_test_framework
4 DEPS =
5
6 #%.o: %.cpp $(DEPS)
7 #     $(CC) $(CFLAGS) -c $<
8
9 all: TextWriter test
10
11 TextWriter: main.o RandWriter.o
12     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
13
14 test: test.o RandWriter.o
15     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
16
17 main.o: main.cpp RandWriter.h
18     $(CC) $(CFLAGS) -c main.cpp $(LIBS)
19
20 RandWriter.o: RandWriter.cpp RandWriter.h
21     $(CC) $(CFLAGS) -c RandWriter.cpp $(LIBS)
22
23 test.o: test.cpp RandWriter.h
24     $(CC) $(CFLAGS) -c test.cpp $(LIBS)
25
26 clean:
27     rm *.o TextWriter
```

10.6 PS6 Main Source

10.6.1 main.cpp

```
1 /*Name: <John Hua>*/
2 /*Course name: <COMP.2040>*/
3 /*Assignment: <ps6>*/
4 /*Instructor's name: <Dr. James Daly>*/
5 /*Date: <4.10.2022>*/
6 /*Sources Of Help: */
7 /*Copyright 2022 <John Hua>
8 /*****
   */
9 #include "RandWriter.h"
10 #include <iostream>
11 #include <string>
12 #include <vector>
13
14 int main(int argc, char* argv[]) {
15     if (argc < 3 || argc > 3) {
16         std::cout << "Invalid command arguments." << std
            ::endl;
17         std::cout << "Correct usage: ./TextWriter <order
            > <length> < input.txt" << std::endl; //
            NOLINT
18         return 0;
19     }
20
21     int k = atoi(argv[1]);
22     int L = atoi(argv[2]);
23     std::string yoho = "yoho fox";
24
25     std::string text;
26     std::string next;
27     while (std::cin >> next) {
28         text += " " + next;
29         next = "";
30     }
31
32     RandWriter a(text, k);
```

```
33
34  std::string temp;
35  std::string test = "gagggagaggcgagaaa";
36  for (int i = 0; i < k; i++)
37      temp.push_back(text[i]);
38
39  std::cout << a.generate(temp, L) << std::endl;
40 }
```

10.7 PS6 RandWriter Source

10.7.1 RandWriter.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps6>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <4.10.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9
10 #include "RandWriter.h"
11 #include <iostream>
12 #include <algorithm>
13 #include <string>
14 #include <vector>
15 #include <map>
16
17 RandWriter::RandWriter(std::string text, int k) {
18     m_order = k;
19     input_text = text;
20
21     srand(time(NULL));
22
23     /* make the alphabet from the input text string
        */
24     static const size_t npos = -1;
25     for (size_t i = 0; i < text.length(); i++) {
26         if (alphabet.find(text[i]) == npos) {
27             alphabet.push_back(text[i]);
28         }
29     }
30     /*sort the kgram table in descending order*/
31     std::sort(alphabet.begin(), alphabet.end(),
32         [](char a, char b) {
33             return a > b;
34         });
35 }
```



```

35
36     /* make the kgram table */
37     for (size_t i = 0; i < text.length(); i++) {
38         std::string tmp;
39         std::string tmp2;
40         for (size_t j = i; j < i + k; j++) {
41             if (j >= text.length()) {
42                 tmp.push_back(text[j - text.size()])
43                 ;
44             } else {
45                 tmp.push_back(text[j]);
46             }
47         }
48         if (kgram_cnt.end() == kgram_cnt.find(tmp))
49             {
50                 kgram_cnt[tmp] = 1;
51             } else {
52                 kgram_cnt[tmp]++;
53             }
54         for (size_t j = 0; j < alphabet.length(); j
55             ++) {
56             if (kgram_cnt.find(tmp + alphabet[j]) ==
57                 kgram_cnt.end())
58                 kgram_cnt[tmp + alphabet[j]] = 0;
59         }
60         for (size_t j = i; j < i + (k + 1); j++) {
61             if (j >= text.length()) {
62                 tmp2.push_back(text[j - text.length
63                     ()]);
64             } else {
65                 tmp2.push_back(text[j]);
66             }
67             kgram_cnt[tmp2]++;
68         }
69     }

```

```

70 /*returns the order of the markov model*/
71 int RandWriter::orderK() const {
72     return m_order;
73 }
74
75 /*returns the frequency of given kgram in the
    sequence*/
76 int RandWriter::freq(std::string kgram) {
77     if (kgram.size() != (unsigned)m_order) {
78         throw std::runtime_error
79             ("Invalid length k for kgram.");
80     }
81
82     std::map<std::string, int>::iterator it;
83     it = kgram_cnt.find(kgram);
84
85     if (it == kgram_cnt.end()) {
86         return 0;
87     } else {
88         return it->second;
89     }
90 }
91
92 /*returns the frequency of given kgram followed by
    the
93 given char 'c' in the sequence. throws an exception
94 if given kgram length != order*/
95 int RandWriter::freq(std::string kgram, char c) {
96     if (kgram.size() != (unsigned)m_order) {
97         throw std::runtime_error
98             ("Invalid length k for kgram.");
99     }
100     if (m_order == 0) {
101         int cnt = 0;
102         for (size_t i = 0; i < input_text.size(); i
            ++){
103             if (input_text[i] == c) {
104                 cnt++;
105             }
106         }

```

```

107         return cnt;
108     } else {
109         return kgram_cnt[kgram + c];
110     }
111 }
112
113 /*returns a random char following the kgram passed
    to kRand
114 throws exception if the kgram length != order or if
    the
115 kgram doesn't exist within the sequence*/
116 char RandWriter::kRand(std::string kgram) {
117     std::map<std::string, int>::iterator it;
118     it = kgram_cnt.find(kgram);
119
120     if (kgram.size() != (unsigned)m_order) {
121         throw std::runtime_error
122             ("Invalid length k for kgram.");
123     } else if (it == kgram_cnt.end()) {
124         throw std::runtime_error
125             ("No such kgram exists");
126     }
127     std::string tmp;
128     for (size_t i = 0; i < alphabet.size(); i++) {
129         for (int j = 0; j < kgram_cnt[kgram +
            alphabet[i]]; j++)
130             tmp.push_back(alphabet[i]);
131     }
132     return tmp[rand() % tmp.size()];
133 }
134
135 /*generates string of L characters given by command
    line argument*/
136 std::string RandWriter::generate(std::string kgram,
    int L) {
137     if (kgram.size() != (unsigned)m_order) {
138         throw std::runtime_error
139             ("Invalid length k for kgram.");
140     }
141     std::string gen_string = kgram;

```

```

142     char rand_c;
143     for (size_t i = 0; i < L - (unsigned)m_order; i
        ++ ) {
144         rand_c = kRand(gen_string.substr(i, m_order)
            );
145         gen_string.push_back(rand_c);
146     }
147     return gen_string;
148 }
149
150 /*outputs the kgram table*/
151 std::ostream& operator<< (std::ostream &output,
    RandWriter &model) {
152     output << "\n_Order: " << model.m_order << "\n";
153     output << "Alphabet: " << model.alphabet << "\n";
154
155     output << "Kgrams Table:" << std::endl;
156
157     std::map<std::string, int>::iterator it;
158     std::map<std::string, int> model_temp = model.
        kgram_cnt;
159
160     for (it = model_temp.begin(); it != model.
        kgram_cnt.end(); it++) {
161         output << it->first << " " << it->second << "\n"
            ;
162     }
163
164     return output;
165 }

```

10.8 PS6 RandWriter Header

10.8.1 RandWriter.h

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps6>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <4.10.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9  #ifndef _HOME_JHUA_COMPUTING_IV_PS6_RANDWRITER_H_
10 #define _HOME_JHUA_COMPUTING_IV_PS6_RANDWRITER_H_
11 #include <iostream>
12 #include <string>
13 #include <vector>
14 #include <map>
15
16 class RandWriter {
17 public:
18     RandWriter(std::string text, int k);
19     /*Order k of Markov model*/
20     int orderK() const;
21     int freq(std::string kgram);
22     int freq(std::string kgram, char c);
23     char kRand(std::string kgram);
24     std::string generate(std::string kgram, int L);
25
26     friend std::ostream& operator<<(std::ostream&
        output, RandWriter& model);
27
28 private:
29     int m_order;
30     std::map <std::string, int> kgram_cnt;
31     std::string input_text;
32     std::string alphabet;
33 };
34
```

```
35 #endif // _HOME_JHUA_COMPUTING_IV_PS6_RANDWRITER_H_
```

10.9 PS6 Boost Source

10.9.1 test.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps6>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <4.10.2022>*/
6  /*Sources Of Help: */
7  /*Copyright 2022 <John Hua>
8  /*****
    */
9  #include <iostream>
10 #include <string>
11 #include <exception>
12 #include <stdexcept>
13
14 #include "RandWriter.h"
15
16 #define BOOST_TEST_DYN_LINK
17 #define BOOST_TEST_MODULE Main
18 #include <boost/test/unit_test.hpp>
19
20 BOOST_AUTO_TEST_CASE(model_zero) {
21     RandWriter model_zero
22     ("gagggagaggcgagaaa", 0);
23
24     // function must return zero
25     BOOST_REQUIRE
26     (model_zero.orderK() == 0);
27
28     // throws excpt if length of kgram != order
29     BOOST_REQUIRE_THROW
30     (model_zero.freq("g"), std::runtime_error);
31
32     // length of sequence
33     BOOST_REQUIRE
34     (model_zero.freq("") == 17);
35 }
```

```

36     // occurrences of 'g' in sequence
37     BOOST_REQUIRE
38     (model_zero.freq("", 'g') == 9);
39
40     // occurrences of 'b' in sequence
41     BOOST_REQUIRE
42     (model_zero.freq("", 'b') == 0);
43
44     BOOST_REQUIRE_THROW
45     (model_zero.kRand("g"), std::runtime_error);
46 }
47
48 BOOST_AUTO_TEST_CASE(model_one) {
49     // initializing test sequence order=1
50     RandWriter model_one
51     ("gagggagagggcgagaaa", 1);
52
53     BOOST_REQUIRE(model_one.orderK() == 1);
54     BOOST_REQUIRE_THROW(model_one.freq("gg"), std::
55         runtime_error);
56     BOOST_REQUIRE_NO_THROW(model_one.freq("g"));
57     BOOST_REQUIRE(model_one.freq("g", 'a') == 5);
58     BOOST_REQUIRE_THROW(model_one.generate("ga", 2),
59         std::runtime_error);
60 }

```


11 PS7: Kronos Time Parsing

11.1 Overview

Assignment PS7 dealt with regular expressions and using Boost's regex library to parse a boot log from a Kronos InTouch time application. Using regular expressions and parsing the log file, we were tasked with verifying the device boot up timings and determining whether the boot was successful or it failed. A boot is successful if a certain given line of text followed a successful boot, and a boot was deemed a failure or incomplete is when another boot up message follows the previous one. We then set out to build a regular expression that can match a successful boot and one that matches an end boot.

11.2 Key Concepts

Key concepts from PS7 included utilizing Boost's regex library to parse and match expressions. We were given a Kronos time log and the text criteria which would account for a successful boot and a boot start-up. We were given lecture notes on how to properly use the date and time libraries to correctly parse the timestamps in the Kronos log and then correctly output it in another log file that contained all the successful and incomplete boots. A successful boot contained a timestamp followed by the text "(log.c.166) server started", and a boot successfully completed if it was followed by a string of text containing a timestamp in addition to the line "INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080". With these two regular expressions in our arsenal, we can then parse the Kronos log file and check for successful boots, and branch to an incomplete boot if we found another boot/start-up from a previous boot-up.

11.3 Learnings

From this assignment I learned how regular expressions and how to build accordingly to match any text or string using Boost regex's library. The framework also provided other useful libraries such as date and time we would probably prove to be useful in other applications in the software field. With these skills, I can use the libraries to parse anything I needed and build an application around it if need be. As with other assignments, PS7 contained a lot of practicality and Computing IV as a whole gave us a introductory into building practical applications on a small scale.

11.4 Output

Figure 11: Image containing a successful parse of a log file and outputting the boot statuses from it

```
=== Device boot ===
435369(device1_intouch.log): 2014-03-25 19:11:59    Boot Start
435759(device1_intouch.log): 2014-03-25 19:15:02    Boot Completed
|
|    Boot Time: 183000ms
|
=== Device boot ===
436500(device1_intouch.log): 2014-03-25 19:29:59    Boot Start
436859(device1_intouch.log): 2014-03-25 19:32:44    Boot Completed
|
|    Boot Time: 165000ms
|
=== Device boot ===
440719(device1_intouch.log): 2014-03-25 22:01:46    Boot Start
440791(device1_intouch.log): 2014-03-25 22:04:27    Boot Completed
|
|    Boot Time: 161000ms
|
=== Device boot ===
440866(device1_intouch.log): 2014-03-26 12:47:42    Boot Start
441216(device1_intouch.log): 2014-03-26 12:50:29    Boot Completed
|
|    Boot Time: 167000ms
|
=== Device boot ===
442094(device1_intouch.log): 2014-03-26 20:41:34    Boot Start
442432(device1_intouch.log): 2014-03-26 20:44:13    Boot Completed
|
|    Boot Time: 159000ms
```

11.5 PS7 Makefile

11.5.1 Makefile

```
1 CC = g++
2 CFLAGS = -Wall -Werror -pedantic --std=c++14
3 LIBS = -lboost_regex
4 DEPS =
5
6 #%.o: %.cpp $(DEPS)
7 #     $(CC) $(CFLAGS) -c $<
8
9 all: ps7
10
11 ps7: main.o
12     $(CC) $(CFLAGS) -o $@ $^ $(LIBS)
13
14 main.o: main.cpp
15     $(CC) $(CFLAGS) -c main.cpp $(LIBS)
16
17 clean:
18     rm *.o ps7
```

11.6 PS7 Main Source

11.6.1 main.cpp

```
1  /*Name: <John Hua>*/
2  /*Course name: <COMP.2040>*/
3  /*Assignment: <ps7a>*/
4  /*Instructor's name: <Dr. James Daly>*/
5  /*Date: <4.21.2022>*/
6  /*Sources Of Help: Dr. Daly's Discord*/
7  /*Copyright 2022 <John Hua>*/
8  /*
   *****
   */
9  #include <iostream>
10 #include <fstream>
11 #include <string>
12
13 #include <boost/regex.hpp>
14 #include <boost/date_time/gregorian/gregorian.hpp>
15 #include <boost/date_time/posix_time/posix_time.hpp>
16
17
18 using boost::gregorian::date;
19 using boost::posix_time::ptime;
20 using boost::posix_time::time_duration;
21
22 int main(int argc, char* argv[]) {
23     std::string fileName = argv[1];
24     std::ifstream log(argv[1]);
25     std::ofstream rpt(fileName + ".rpt", std::
        ofstream::out);
26     std::string line;
27
28     if (argc < 2 || argc > 2) {
29         throw std::runtime_error
30             ("Correct usage: ./ps7 <log_file_name>");
31     }
32
33 }
```

```

34     time_duration bootTime;
35
36     date startDate;
37     date endDate;
38
39     ptime startTime;
40     ptime endTime;
41
42     int lineCnt = 1;
43
44     boost::regex startBoot(
45         "([0-9]{4})-([0-9]{2})-([0-9]{2}) "
46         "([0-9]{2}):([0-9]{2}):([0-9]{2}): "
47         "\\(log.c.166\\) "
48         "server started.*");
49
50     boost::regex endBoot(
51         "([0-9]{4})-([0-9]{2})-([0-9]{2}) "
52         "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}) "
53         ":INFO:oejs.AbstractConnector:Started "
54         "SelectChannelConnector@0.0.0.0:9080.*");
55
56     bool consequBoot = false;
57     if (log.is_open()) {
58         while (getline(log, line)) {
59             boost::smatch sm;
60             if (boost::regex_match(line, sm,
61                                     startBoot)) {
62                 startDate = date(stoi(sm[1]), stoi(
63                                     sm[2]), stoi(sm[3]));
64                 startTime = ptime
65                     (startDate, time_duration
66                     (stoi(sm[4]), stoi(sm[5]), stoi(sm
67                                     [6]))));
68
69                 /*Found consecutive boot line following
70                 start boot*/
71                 if (consequBoot) {
72                     rpt << "**** Incomplete Boot ****\n"
73                         << std::endl;

```

```

69         conseqBoot = false;
70     }
71     std::string dateString = sm[1] + "-" +
        sm[2] + "-" + sm[3];
72     std::string timeString = sm[4] + ":" +
        sm[5] + ":" + sm[6];
73     rpt << "=== Device boot ===\n"
74         << lineCnt << "(" << fileName << "):
        "
75         << dateString << " "
76         << timeString
77         << "      Boot Start" << std::endl;
78     conseqBoot = true;
79
80     /*Found successful boot line following
        start boot*/
81     } else if (boost::regex_match(line, sm,
        endBoot)) {
82         endDate = date(stoi(sm[1]), stoi(sm
            [2]), stoi(sm[3]));
83         endTime = ptime
84             (endDate, time_duration(stoi(sm[4]),
            stoi(sm[5]), stoi(sm[6])));
85         std::string dateString = sm[1] + "-" +
            sm[2] + "-" + sm[3];
86         std::string timeString = sm[4] + ":" +
            sm[5] + ":" + sm[6];
87         rpt << lineCnt << "(" << fileName << "):
            "
88         << dateString << " "
89         << timeString
90         << "      Boot Completed" << std::endl
            ;
91         bootTime = endTime - startTime;
92         rpt << "      Boot Time: "
93         << bootTime.total_milliseconds() <<
            "ms\n" << std::endl;
94         conseqBoot = false;
95     }
96     lineCnt++;

```

```
97         }
98         rpt.close();
99         log.close();
100     }
101 }
```