

Likhitha Shrinivas Gudalwar

COMP IV: Project Portfolio

10<sup>th</sup> December Fall 2021

**Contents:**

**PS0 Hello World with SFML**

**PS1a Linear Feedback Shift Register and Image Encoding:** Linear Feedback Shift Register

**PS1b Linear Feedback Shift Register and Image Encoding:** PhotoMagic

**PS2a N-Body Simulation:** Loading universe files; body class; graphics

**PS2b N-Body Simulation**

**PS3 Recursive Graphics**

**PS4a Synthesizing a Plucked String Sound:** CircularBuffer implementation

**PS4b Synthesizing a Plucked String Sound:** StringSound implementation and SFML audio output

**PS5 DNA Sequence Alignment**

**PS6 Random Writer**

**PS7 Kronos Time Clock**

**Time to complete:** 12 hours

## **PS0: Hello World with SFML**

### **About the Assignment**

Our first Computing IV assignment was Hello World. This assignment's main goal was to set up our Linux build environment and test the SFML audio/graphics library. This included installing Linux – either via a Virtual Box image or natively – and running some SFML example code to test SFML. To make the demo code do something interesting, we had to extend it. I was already familiar with Linux at this point, so it didn't take long to set up my environment – a few `sudo apt-get install` commands later, and I had SFML ready to go.

### **Key Concepts**

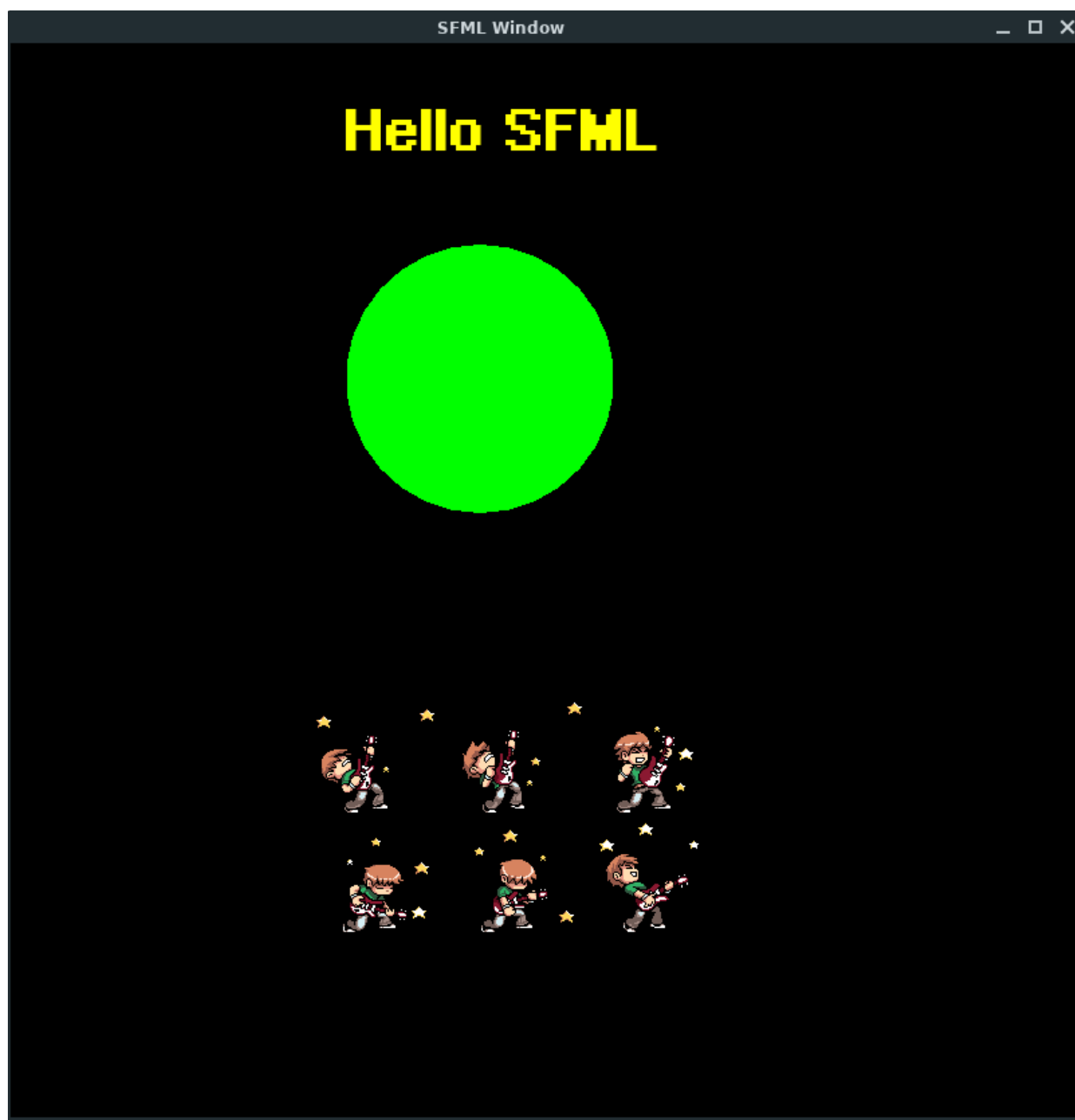
Because this was my first assignment, I didn't use any algorithms to build the program. Mostly just basic programming stuff like variables, objects, and a couple of while / for loops to keep the window open and the sprites moving around. After all, the main purpose of this assignment was to experiment with SFML's various classes, which included Images, Sprites, Text, Texture, and Keyboard. Texture, Image, and Sprite classes all interact with one another – an image must be loaded into a Texture, and a sprite must set a Texture to be used.

For this assignment, I wrote a program that displays a moving image on an SFML window using SFML's sprite class. To move it around, I added some basic controls i.e the arrow keys for up/down/left/right. I also used the keyboard interactions in which clicking the escape on keyboard will quit the sfml window. Apart from this I used the position of the sprites to detect the collisions in between the sprites.

### **What I Learned**

I learned a few things about sfml while working on this assignment. I learned how to use SFML at a basic level, how to display images in an SFML window, and even how to control sprites using SFML's Keyboard library. Because I was already familiar with Linux, working on this wasn't too difficult, and things like installing SFML were simple because all I had to do was boot Linux up and run `sudo apt-get install` the required sfml.

## Screenshot



## **Source code for PS0 Hello World with SFML**

### **Makefile**

```
1: CXX = g++
2: CXXFLAGS = -Wall -Werror -ansi -pedantic
3: LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
4:
5:
6: all:
7:      $(CXX) -c main.cpp $(CXXFLAGS)
8:      $(CXX) main.o -o sfml-app $(LFLAGS)
9: clean:
10:      rm main.o
11:      rm sfml-app
12:
```

**main.cpp**

```
1: #include<SFML/Graphics.hpp>
2: #include<iostream>
3: using namespace std;
4: int main()
5: {
6:     //Create the main window
7:     sf::RenderWindow window(sf::VideoMode(900, 900), "SFML Window");
8:
9:     //sets framerate limit
10:    window.setFramerateLimit(120);
11:
12:    //Creates a SFML shape
13:    sf::CircleShape shape(100.f);
14:
15:    //fills the shape with the color indicated
16:    shape.setFillColor(sf::Color::Green);
17:
18:    //sets the position of the shape
19:    shape.setPosition(250,150);
20:
21:    // Load a sprite to display
22:    sf::Texture texture;
23:
24:    if (!texture.loadFromFile("sprite.png"))
25:        //exits the code giving an error message if image doesn't exist
26:        return EXIT_FAILURE;
27:
28:    //Create SFML sprite
29:    sf::Sprite sprite(texture);
30:
31:    //sets the position of the sprite
32:    sprite.setPosition(sf::Vector2f(250,400));
33:
34:    //Create a graphical text to display
35:    sf::Font font;
36:    if (!font.loadFromFile("PIXEAB__.TTF"))
37:        return EXIT_FAILURE;
38:    sf::Text text("Hello SFML", font, 30);
39:
40:
41:    //sets the position of the text
42:    text.setPosition(250,50);
43:
44:    //displays the text in assigned color
45:    text.setFillColor(sf::Color::Yellow);
46:
47:    //Start the window loop
48:    while (window.isOpen())
49:    {
50:        //Process Events
51:        sf::Event event;
52:
53:        //Event handler loop
54:        while (window.pollEvent(event))
```

```

55:         {
56:             //Close window:exit
57:             if (event.type == sf::Event::Closed)
58:                 window.close();
59:         }
60:
61: if(sf::Keyboard::isKeyPressed(sf::Keyboard::A)&&
    sprite.getPosition().x>0)
62:     {
63:         //moves the sprite image to the left on pressing the key A
64:         sprite.move(-1,0);
65:     }
66:
67: if(sf::Keyboard::isKeyPressed(sf::Keyboard::W) &&
    sprite.getPosition().y>0)
68:     {
69:         //moves the sprite image above on pressing the key W
70:         sprite.move(0,-1);
71:     }
72: if(sf::Keyboard::isKeyPressed(sf::Keyboard::S)&& sprite.getPosition().y +
73:     sprite.getGlobalBounds().height < window.getSize().y) {
74:         //moves the sprite image down on pressing the key S
75:         sprite.move(0,1);
76:     }
77: if(sf::Keyboard::isKeyPressed(sf::Keyboard::D)&& sprite.getPosition().x
78:     + sprite.getGlobalBounds().width < window.getSize().x)
79:     {
80:         //moves the isprite image to the right on pressing the key D
81:         sprite.move(1,0);
82:     }
83:
84:         //detects whether the sprite collides with the shape or not
85:
86: if(sprite.getGlobalBounds().intersects(shape.getGlobalBounds()))
87:     {
88:         //prints if collision occurs
89:         cout<<"Collision occurs between Shape and Sprite"<<endl;
90:     }
91:     else
92:     {
93:         //prints if no collision occurs
94:         cout<<"No collision occurs between Shape and Sprite"<<endl;
95:     }
96:
97:         //closes the SFML viewer when esc key is pressed
98: if (event.key.code == sf::Keyboard::Escape)
99:     window.close();
100:
101:         //Clear the window screen
102: window.clear();
103:
104:         //draw the shape
105: window.draw(shape);
106:
107:         //draw the sprite image
108: window.draw(sprite);

```

```
109:
110:     //draws text
111:     window.draw(text);
112:
113:     //update the window
114:     window.display();
115: }
116:
117: return EXIT_SUCCESS;
118: }
```

## **PS1a: Linear Feedback Shift Register**

### **About the Assignment**

We had to create a Linear Feedback Shift Register for this project. This register shifts all bits left one position and then XORs the leftmost bit and the seed bit to fill the empty space on the far-right side of the register after the shift. Our main goal was to implement the shift register in a class called "FibLFSR" and to use the Boost test framework to write several unit tests.

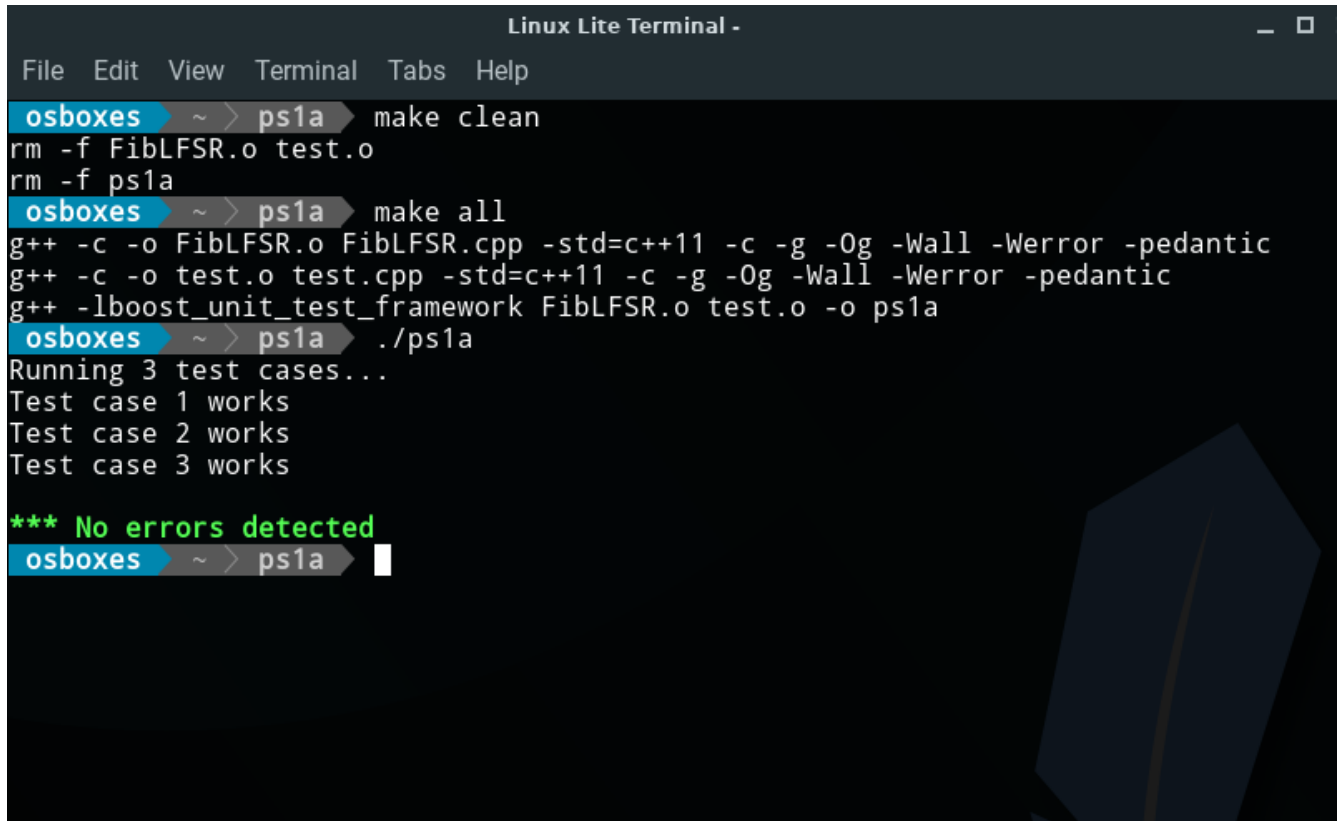
### **Key Concepts**

I used the Boost test framework for this project to test our FibLFSR class, which in my code represents the shift register as a C++ string. I implemented the step and generate methods using bitset as the representation for register bits. Shifting left was accomplished by the step function by appending the result of XORing the left most bit with the tap position. The generate function determines the number of times the step is to be performed on the register bits. The Boost test framework was used to test our FibLFSR class, by using Boost's auto test case methods to test the step / generate methods against given test cases.

### **What I Learned**

This assignment taught me a lot about C++ testing. I've only ever studied testing in my software engineering class, but I've never actually used it in my code. In the past, I've done a combination of compiling, ensuring that the program runs, and then manually testing different aspects to see if they work properly. Using the Boost test framework simplified things – I could write tests, change my code, and know with a few commands to detect which part of code is working and which is not. In the future, I intend to continue using unit testing for projects that become complex or difficult to manually test on a regular basis.



**Screenshot:**

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
osboxes ~ > ps1a make clean
rm -f FibLFSR.o test.o
rm -f ps1a
osboxes ~ > ps1a make all
g++ -c -o FibLFSR.o FibLFSR.cpp -std=c++11 -c -g -Og -Wall -Werror -pedantic
g++ -c -o test.o test.cpp -std=c++11 -c -g -Og -Wall -Werror -pedantic
g++ -lboost_unit_test_framework FibLFSR.o test.o -o ps1a
osboxes ~ > ps1a ./ps1a
Running 3 test cases...
Test case 1 works
Test case 2 works
Test case 3 works

*** No errors detected
osboxes ~ > ps1a
```

**Source code for PS1a Linear Feedback Shift Register:****Makefile:**

```
1: CXX=g++
2: CXXFLAGS=-std=c++11 -c -g -Og -Wall -Werror -pedantic
3: LDLIBS=-lboost_unit_test_framework
4: LDFLAGS=$(LDLIBS)
5:
6: OBJS= FibLFSR.o test.o
7:
8:
9: all: ps1a
10:
11: ps1a: FibLFSR.o test.o
12:      $(CXX) $(LDFLAGS) $(OBJS) -o ps1a
13:
14: FibLFSR.o : FibLFSR.cpp FibLFSR.h
15:      $(CXX) -c -o FibLFSR.o FibLFSR.cpp $(CXXFLAGS)
16:
17: test.o: test.cpp
18:      $(CXX) -c -o test.o test.cpp $(CXXFLAGS)
19:
20: clean:
21:      rm -f $(OBJS)
22:      rm -f ps1a
23:
```

**FibLFSR.cpp:**

```

1: #include "FibLFSR.h"
2: #include<string>
3: #include<iostream>
4: using namespace std;
5:
6: //constructor of class FibLFSR
7: FibLFSR::FibLFSR(string seed)
8: {
9: //to check if size of register is greater than 16 or not
10: if(seed.length()>16)
11: {
12: //if greater displays this statement
13: cout<<"Seed cannot be more than 16 bits"<<endl;
14: }
15: for(size_t i=0 ;i<seed.length();i++)
16: {
17: char bit{seed[i]};
18:
19: if(bit !='1' && bit !='0')
20: {
21: //if bits of register contains digits other than 0 or 1 displays the
22: // below statement
23: cout<<"your register must contain 0's and 1's"<<endl;
24: }
25:
26: //sets the index value (15-i) to bit value '1'
27: reg.set(15 - i, bit =='1');
28:
29: }
30: }
31:
32: int FibLFSR ::step()
33: {
34:
35: //performs X-OR operation for the 15th, 13th, 12th, 10th register bits
36: int step_output = reg[15] ^ reg[13] ^ reg[12] ^ reg[10] ;
37:
38: //shifts the bits of register to the left
39: reg <<= 1;
40:
41: //sets the 0th bit of the register with the output bit
42: reg.set(0, static_cast<bool>(step_output));
43:
44: //returns value of the output_bit
45: return step_output;
46:
47: }
48:
49: int FibLFSR:: generate(int k)
50: {
51: uint16_t generate_output=0;
52: //performs step operation k times
53: for(int i=0;i<k;i++)
54: {

```

```

55: //shifts the generate_output register to the left
56: generate_output <<=1;
57: //performs OR operation in between the bits left shifted and the
58: step_output
59: generate_output |= step();
60:
61: }
62: return generate_output;
63: }
64:
65: std::ostream& operator<<(std::ostream& os, const FibLFSR lfsr)
66: {
67: // Since operator<< is a friend of the FibLFSR class, we can access
68: // FibLFSR class members directly.
69: std::string bits;
70: //
71: bits.reserve(FibLFSR::REG_SIZE);
72: for(int i=(FibLFSR::REG_SIZE -1); i>=0;i--)
73: {
74: //pushes bits into the register for every iterative step
75: bits.push_back('0'+lfsr.reg[i]);
76: }
77: return os<<bits; // return std::ostream so we can chain calls to
78: // operator<<

79: }
80: //Destructor of the class FibLFSR
81: FibLFSR::~FibLFSR()
82: {
83:
84: }

```

**FibLFSR.h:**

```
1: #ifndef FIBLFSR_H
2: #define FIBLFSR_H
3:
4: #include<string>
5: #include<bitset>
6: #include<iostream>
7: using namespace std;
8:
9: class FibLFSR
10: {
11:
12: public:
13: //costructor of FibLFSR class
14: FibLFSR(string seed); //seed is a 16bits long string
15:
16: //performs left shift and xor operation to the register bits
17: int step();
18:
19: //generates a random integer
20: int generate(int k);
21:
22: // std::ostream is the type for object std::cout
23: friend std::ostream& operator<< (std::ostream& os, const FibLFSR lfsr);
24:
25: ~FibLFSR();
26:
27: private:
28:
29: //initializing the register size to 16bits
30: static const size_t REG_SIZE{16};
31:
32: bitset<REG_SIZE> reg{};
33:
34: };
35: #endif
```

**Test.cpp**

```

1:
2: #include <iostream>
3: #include <string>
4: #include "FibLFSR.h"
5:
6: #define BOOST_TEST_DYN_LINK
7: #define BOOST_TEST_MODULE Main
8: #include <boost/test/included/unit_test.hpp>
9:
10: BOOST_AUTO_TEST_SUITE (FIBONACCI_LFSR)
11: //FIBONACCI_LFSR is the test suite name
12: //This test case is for the 16bit lfsr register
13:
14: BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps1) {
15:
16: //In this phase, we create a string and on which the functions will work
17: //Here the LFSR register contains the string "1011011000110110"
18:
19:     FibLFSR l("1011011000110110");
20:
21: //this function checks whether the resulted seed bits of the string
22: // provided matches the one in the unit test case
23: //this phase makes our unit test pass or fail
24:     BOOST_REQUIRE(l.step() == 0);
25:     BOOST_REQUIRE(l.step() == 0);
26:     BOOST_REQUIRE(l.step() == 0);
27:     BOOST_REQUIRE(l.step() == 1);
28:     BOOST_REQUIRE(l.step() == 1);
29:     BOOST_REQUIRE(l.step() == 0);
30:     BOOST_REQUIRE(l.step() == 0);
31:     BOOST_REQUIRE(l.step() == 1);
32:     BOOST_REQUIRE(l.step() == 1);
33:     BOOST_REQUIRE(l.step() == 0);
34:
35:
36: //this function checks whether the resulted generate output is same as
37: // the one in the unit test case
38:     FibLFSR l2("1011011000110110");
39:
40: //this phase makes our unit test pass or fail
41:     BOOST_REQUIRE(l2.generate(9) == 51);
42:     std::cout<<"Test case 1 works"<<endl;
43:
44: }
45:
46: //This test case is for the 16bit lfsr register
47: BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps2)
48:
49: {
50: //In this phase, we create a string and on which the functions will work
51: //Here the LFSR register contains the string "1110101110111011"
52:
53:     FibLFSR l("1110101110111011");
54:

```

```

55: //this function checks whether the resulted seed bits of the string
56: // provided matches the one in the unit test case
57: //this phase makes our unit test pass or fail
58: BOOST_REQUIRE(l.step() == 0);
59: BOOST_REQUIRE(l.step() == 1);
60: BOOST_REQUIRE(l.step() == 1);
61: BOOST_REQUIRE(l.step() == 0);
62: BOOST_REQUIRE(l.step() == 1);
63: BOOST_REQUIRE(l.step() == 1);
64: BOOST_REQUIRE(l.step() == 1);
65: BOOST_REQUIRE(l.step() == 1);
66: BOOST_REQUIRE(l.step() == 1);
67: BOOST_REQUIRE(l.step() == 1);
68:
69: //this function checks whether the resulted generate output is same as
70: // the one in the unit test case
71: FibLFSR l2("1110101110111011");
72:
73: //this phase makes our unit test pass or fail
74:
75: BOOST_REQUIRE(l2.generate(9) == 223) ;
76:     std::cout<<"Test case 2 works"<<endl;
77: }
78:
79: //This test case is for the 16bit lfsr register
80:
81: BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps3) {
82:
83: //In this phase, we create a string and on which the functions will work
84:
85: //Here the LFSR register contains the string "0110110100110100"
86: FibLFSR l("0110110100110100");
87:
88: //this function checks whether the resulted seed bits of the string
89: // provided matches the one in the unit test case this phase makes our
90: // unit test pass or fail
91: BOOST_REQUIRE(l.step() == 0);
92: BOOST_REQUIRE(l.step() == 0);
93: BOOST_REQUIRE(l.step() == 0);
94: BOOST_REQUIRE(l.step() == 1);
95: BOOST_REQUIRE(l.step() == 0);
96: BOOST_REQUIRE(l.step() == 1);
97: BOOST_REQUIRE(l.step() == 1);
98: BOOST_REQUIRE(l.step() == 0);
99: BOOST_REQUIRE(l.step() == 1);
100: BOOST_REQUIRE(l.step() == 1);
101: //this function checks whether the resulted generate output is same as
102: // the one in the unit test case
103: FibLFSR l2("0110110100110100");
104:
105: //this phase makes our unit test pass or fail
106:
107: BOOST_REQUIRE(l2.generate(9) == 45);
108:     std::cout<<"Test case 3 works"<<endl;
109:
110: }
111:

```

```
112: BOOST_AUTO_TEST_SUITE_END( )  
113:
```



## **PS1b: Image Encoding**

### **About the Assignment**

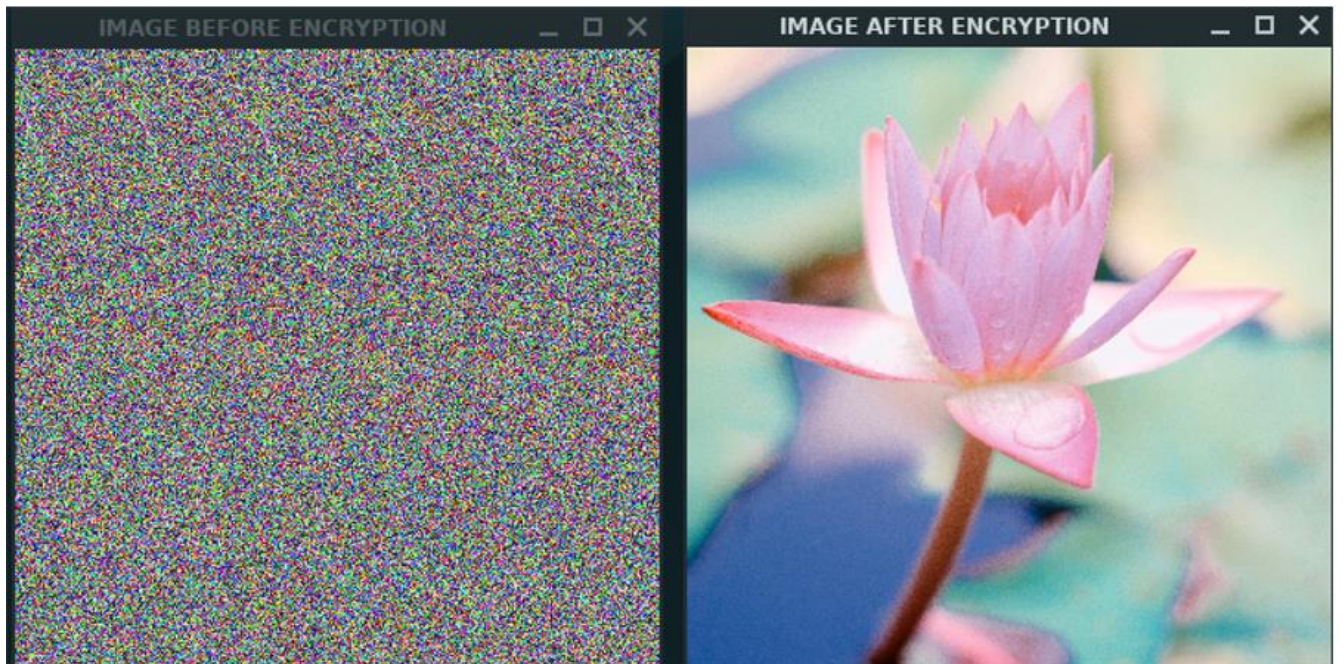
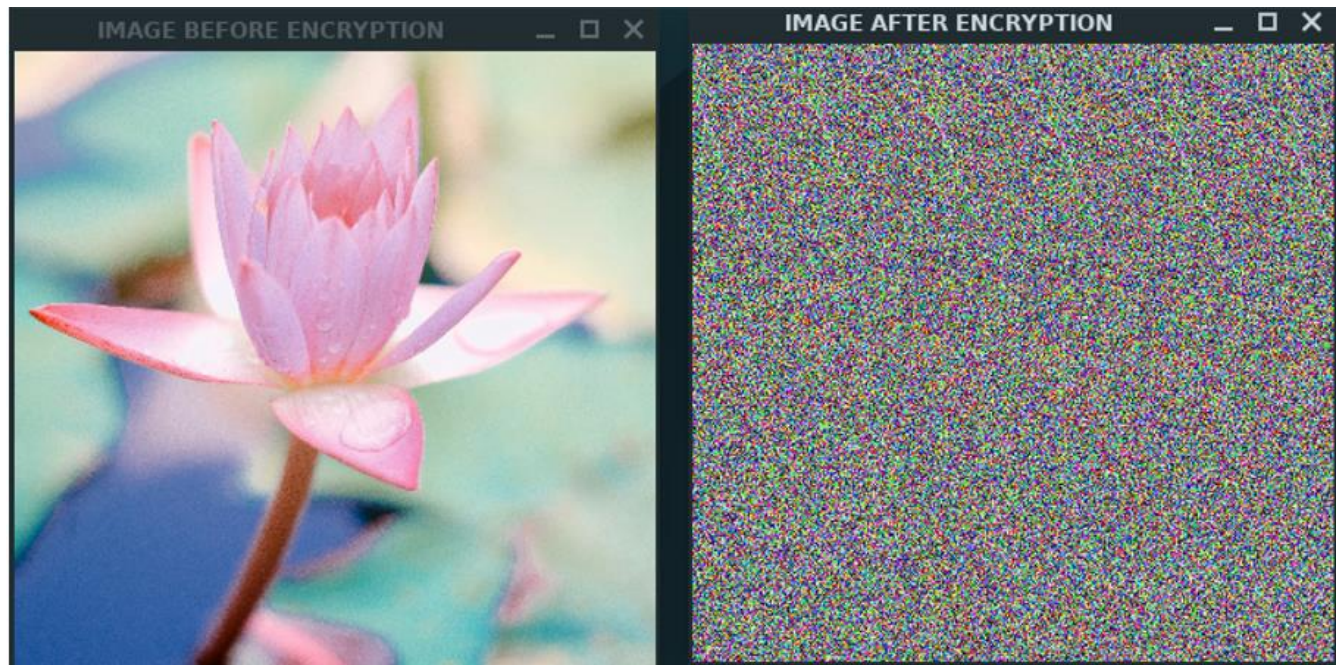
This assignment was an extension of the previous assignment (PS1a). Using the FibLFSR class that we built in PS1a, we were asked to write a program that reads a photo as input from the command line and then outputs the same image, but in encrypted form i.e an encoded image. The FibLFSR class was used to encode the image by left shifting all the bits in the image – thus encoding it using XOR. We also had to display the image to an SFML window and save the encrypted image to a file. And also, the decrypting process that decodes the encoded image and gives as output the original image.

### **Key Concepts**

The FibLFSR class from the previous project was the primary focus of this assignment. In this section, I created a main file PhotoMagic that read three command-line arguments: source image filename, output image filename, and FibLFSR seed. In addition, I used SFML objects such as textures, images, and sprites to read the file, encode it, and output the final encoded image to both the screen and the disk. I used the transform function to encode the image using the function getPixel(), which performs an X-OR operation between the red, green, blue color components and an 8-bit integer generated by the FibLFSR class's generate function.

### **What I Learned**

This project was a lot of fun to work on because it made use of materials that I already had. It was kind of cool to make something for one homework assignment and then reuse it for another. However, because it uses the same FibLFSR class, there wasn't anything new to learn. The encoding and decoding portion of this assignment, on the other hand, proved to be the most beneficial. It was a lot of fun playing with pixels and XORing them to create an encoded image, then displaying the final encoded image. I mostly discovered new applications for the FibLFSR class, which was a fun learning experience because it allowed me to reuse old code for a new purpose.

**Screenshot:**

## **Source code for PS1b Image Encoding**

### **Makefile:**

```
1: CXX=g++
2: CXXFLAGS=-Wall -Werror -pedantic
3: OBJS= FibLFSR.o PhotoMagic.o
4: LIBS= -lsfml-graphics -lsfml-window -lsfml-system
5: LDFLAGS= $(LIBS)
6:
7:
8: all: PhotoMagic
9:
10: PhotoMagic: FibLFSR.o PhotoMagic.o
11:          $(CXX) $(OBJS) -o PhotoMagic $(LDFLAGS)
12:
13: FibLFSR.o : FibLFSR.cpp FibLFSR.h
14:          $(CXX) -c -o FibLFSR.o FibLFSR.cpp $(CXXFLAGS)
15:
16: PhotoMagic.o: PhotoMagic.cpp
17:          $(CXX) -c -o PhotoMagic.o PhotoMagic.cpp $(CXXFLAGS)
18:
19: clean:
20:          rm -f $(OBJS)
21:          rm -f PhotoMagic
22:
```

**PhotoMagic.cpp**

```

1: #include "FibLFSR.h"
2: #include <SFML/System.hpp>
3: #include <SFML/Window.hpp>
4: #include <SFML/Graphics.hpp>
5: #include <iostream>
6:
7: void transform( sf::Image& image, FibLFSR* reg);
8:
9: int main(int argc, char* argv[])
10: {
11:
12: if(argc != 4)
13: {
14: std::cout << "Please follow the executable format ./PhotoMagic
      [input file] [output file] [seed]\n";
15: return -1;
16: }
17:
18: // Save the command line arguments to variables
19: std::string input_file(argv[1]);
20: std::string output_file(argv[2]);
21: std::string seed(argv[3]);
22:
23: FibLFSR lfsr{seed};
24:
25: sf::Image image;
26:
27: //loads image from the file
28: if (!image.loadFromFile(input_file))
29: return EXIT_FAILURE;
30:
31: //gets the size of the image
32: sf::Vector2u size = image.getSize();
33:
34: //creates a window and sets the window size to the same size as image
35: sf::RenderWindow window1(sf::VideoMode(size.x,size.y),"IMAGE BEFORE
      ENCRYPTION");
36: sf::RenderWindow window2(sf::VideoMode(size.x,size.y),"IMAGE AFTER
      ENCRYPTION");
37:
38: //loads an image to display
39: sf::Texture texture;
40: texture.loadFromImage(image);
41:
42: //loads the sprite to display
43: sf::Sprite spritel;
44: spritel.setTexture(texture);
45:
46: //function call to transform function
47: transform(image,&lfsr);
48:

```

```

49: //loads the image file
50: sf::Texture texture2;
51: texture2.loadFromImage(image);
52:
53: //loads the sprite to display
54: sf::Sprite sprite2;
55: sprite2.setTexture(texture2);
56:
57: //starts the window loop
58: while(window1.isOpen() && window2.isOpen())
59: {
60:
61: //process events
62: sf::Event event;
63: while (window1.pollEvent(event))
64: {
65: //close window exit
66: if (event.type == sf::Event::Closed)
67: window1.close();
68: }
69:
70: //clears the screen
71: window1.clear(sf::Color::White);
72:
73: //draws the sprite
74: window1.draw(sprite1);
75:
76: //updates the window
77: window1.display();
78:
79: //process events
80: sf::Event event2;
81: while (window2.pollEvent(event2))
82: {
83: //close window exit
84: if (event2.type == sf::Event::Closed)
85: window2.close();
86: }
87:
88: //clears the screen
89: window2.clear(sf::Color::White);
90:
91: //draws the sprite
92: window2.draw(sprite2);
93:
94: //updates the window
95: window2.display();
96:
97: }
98:
99: //saves the encrypted image into the given filename
100: if (!image.saveToFile(output_file))
101: return -1;
102:
103: return 0;
104:
105: }

```

```
106: void transform( sf::Image& image, FibLFSR* reg)
107: {
108:
109: sf::Color p;
110:
111: for (int x = 0; x < (int)image.getSize().x ; x++)
112: {
113: for (int y = 0; y < (int)image.getSize().y; y++)
114: {
115: //gets the pixels of the image given as input
116: p = image.getPixel(x,y);
117:
118:
119: int r_gen=reg->generate(8);
120: //X-OR operation between the red,green,blue components of color
121: // and an 8-bit integer generated from the generate function
122: p.r = p.r ^ r_gen;
123:
124: p.g = p.g ^ r_gen;
125:
126: p.b = p.b ^ r_gen;
127:
128: //sets the pixel value to the image
129: image.setPixel(x, y, p);
130: }
131:
132: }
133:
134: }
```

**FibLFSR.h**

```
1: #include<string>
2: #include<bitset>
3: #include<iostream>
4: using namespace std;
5:
6: class FibLFSR
7: {
8:
9: public:
10: //costructor of FibLFSR class
11: FibLFSR(string seed); //seed is a 16bits long string
12:
13: //performs left shift and xor operation to the register bits
14: int step();
15:
16: //generates a random integer
17: int generate(int k);
18:
19: // std::ostream is the type for object std::cout
20: friend std::ostream& operator<< (std::ostream& os, const FibLFSR lfsr);
21:
22: ~FibLFSR();
23:
24: private:
25:
26: //initializing the register size to 16bits
27: static const size_t REG_SIZE{16};
28:
29: bitset<REG_SIZE> reg{};
30:
31: };
32:
```

**FibLFSR.cpp**

```

1: #include "FibLFSR.h"
2: #include<string>
3: #include<iostream>
4: using namespace std;
5:
6: //constructor of class FibLFSR
7: FibLFSR::FibLFSR(string seed)
8: {
9: //to check if size of register is greater than 16 or not
10: if(seed.length()>16)
11: {
12: //if greater displays this statement
13: cout<<"Seed cannot be more than 16 bits"<<endl;
14: }
15: for(size_t i=0 ;i<seed.length();i++)
16: {
17: char bit{seed[i]};
18:
19: if(bit !='1' && bit !='0')
20: {
21: //if bits of register contains digits other than 0 or 1 displays the
22: // below statement
23: cout<<"your register must contain 0's and 1's"<<endl;
24: }
25:
26: //sets the index value (15-i) to bit value '1'
27: reg.set(15 - i, bit =='1');
28:
29: }
30: }
31:
32: int FibLFSR ::step()
33: {
34:
35: //performs X-OR operation for the 15th, 13th, 12th, 10th register bits
36: int step_output = reg[15] ^ reg[13] ^ reg[12] ^ reg[10] ;
37:
38: //shifts the bits of register to the left
39: reg <<= 1;
40:
41: //sets the 0th bit of the register with the output bit
42: reg.set(0, static_cast<bool>(step_output));
43:
44: //returns value of the output_bit
45: return step_output;
46:
47: }
48:
49: int FibLFSR:: generate(int k)
50: {
51: uint16_t generate_output=0;
52: //performs step operation k times
53: for(int i=0;i<k;i++)
54: {

```



```

55: //shifts the generate_output register to the left
56: generate_output <<=1;
57: //performs OR operation in between the bits left shifted and the
58: // step_output
59: generate_output |= step();
60:
61: }
62: return generate_output;
63: }
64:
65: std::ostream& operator<<(std::ostream& os, const FibLFSR lfsr)
66: {
67: // Since operator<< is a friend of the FibLFSR class, we can access
68: // FibLFSR
69: std::string bits;
70: //
71: bits.reserve(FibLFSR::REG_SIZE);
72: for(int i=(FibLFSR::REG_SIZE -1); i>=0;i--)
73: {
74: //pushes bits into the register for every iterative step
75: bits.push_back('0'+lfsr.reg[i]);
76: }
77: return os<<bits; // return std::ostream so we can chain calls to
78: // operator<<
79: }
80: FibLFSR::~~FibLFSR()
81: {
82:
83: }
84:

```

## **PS2a: N-Body Simulation: Loading universe files; body class; graphics**

### **About the Assignment**

I worked on the N-Body simulation problem for this assignment. It aims to simulate the universe on a two-dimensional plane using Newton's laws of gravity. It reads two command line arguments – total simulation time and time step – and then displays a static universe on the screen. PS2b was used to implement the finished, moving universe. This section of the assignment was primarily concerned with reading a file from standard I/O and using the data from that file to populate sprites (displaying the various planets) in the correct location in an SFML window.

### **Key Concepts**

We used a few key C++ concepts for this assignment. The first was to read a file into standard I/O using the command line operator. Inside the main program, I simply used `cin` to read the contents of the file. I also used overloading operator `>>` to make it easier to read in data – this way, we could simply type: `cin >> *tmp;`

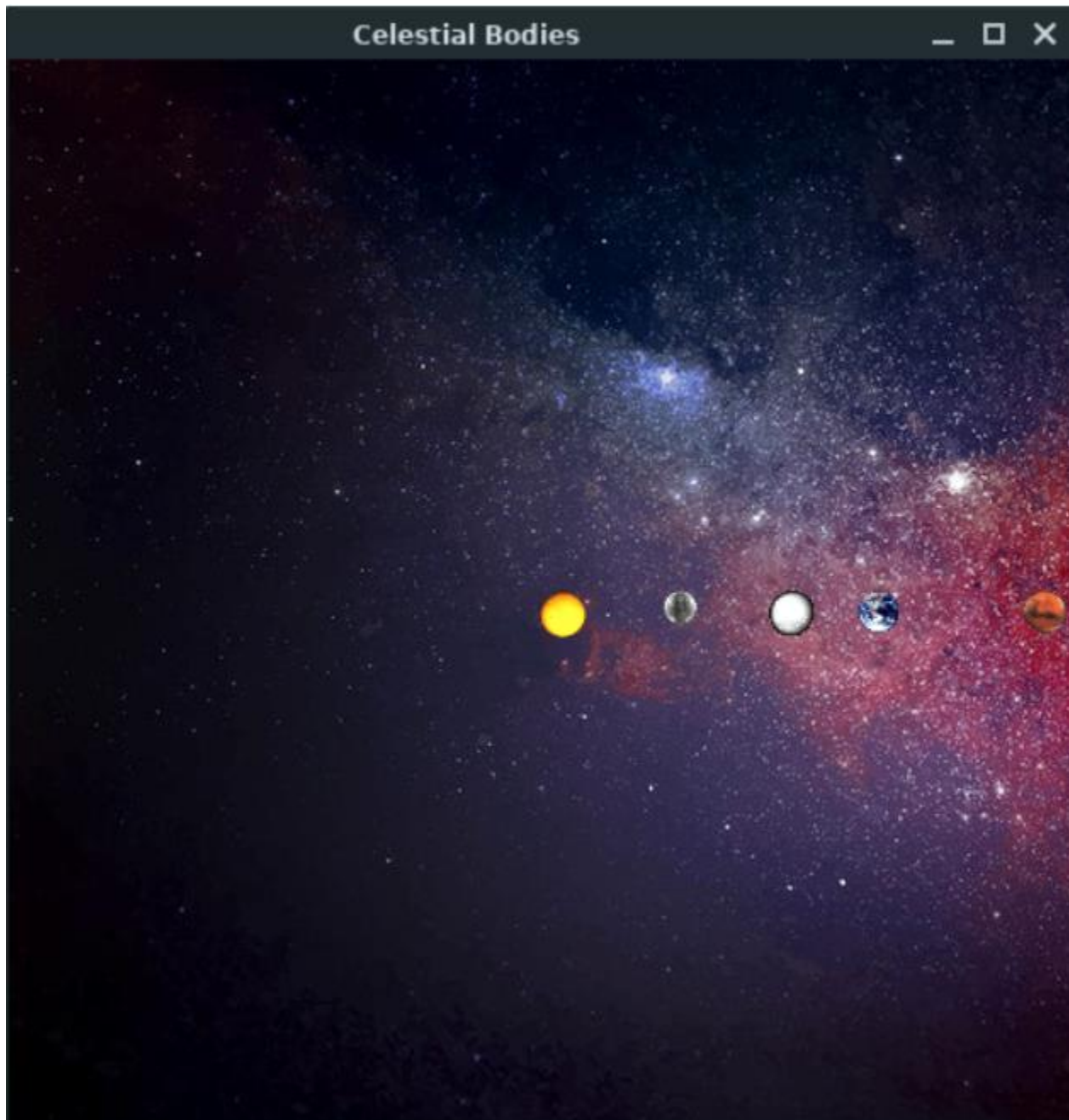
I also used the overloading operator `<<` to output all the data inside the Celestial body object to standard I/O, as shown below: `cout<< *tmp ;`

Another thing I had to do was write a method for computing the planets' X/Y positions and converting them to SFML coordinates. I was able to accomplish this by recognizing that the SFML window system defaults to (0, 0) for the top left corner. To convert the coordinates to the SFML system, I amplified the coordinate by half the height or side of the given window.

### **What I Learned**

In the CelestialBody class, I implemented the `drawUniverse` method, which loads an image to create a universe and iterates through the bodies until each celestial body is displayed. Working on this assignment was quite interesting because it was a visual representation of the celestial bodies, and experimenting with the draw method was a lot of fun. I didn't learn much by using the overloading operators, `>>`, but I hadn't used them in a while, so it was nice to work on something familiar.

Screenshot:



**Source code for PS2a Loading universe files; body class; graphics:****Makefile:**

```
1: CXX= g++
2: CXXFLAGS= -Wall -Werror -pedantic
3: LIBS= -lsfml-graphics -lsfml-window -lsfml-system
4: LDFLAGS= $(LIBS)
5: OBJS= main.o NBody.o
6:
7: all: NBody
8:
9: NBody: main.o NBody.o
10:      $(CXX) $(OBJS) -o NBody $(LDFLAGS)
11:
12: main.o: main.cpp NBody.h
13:      $(CXX) -c main.cpp NBody.h $(CXXFLAGS)
14:
15: NBody.o: NBody.cpp NBody.h
16:      $(CXX) -c NBody.cpp NBody.h $(CXXFLAGS)
17:
18: clean:
19:      rm -f $(OBJS)
20:      rm -f NBody.h.gch
21:      rm -f NBody
22:
```

**Main.cpp**

```

1: #include "NBody.h"
2:
3: int main(int argc, char* argv[]) {
4:     Universe universe;
5:     // function call to readinput
6:     universe.readInput();
7: }
8:
9: void Universe::readInput() {
10:     std::string radius; // NOLINT
11:     std::string num_planets; // NOLINT
12:
13:     // Use cin to redirect the input
14:     std::cin >> num_planets;
15:     std::cin >> radius;
16:
17:     // Converts these from strings to int / float
18:     number_planets = atoi(num_planets.c_str());
19:     universe_radius = atof(radius.c_str());
20:     std::cout << "Number of planets: " << number_planets << std::endl;
21:     std::cout << "Universe Radius: " << universe_radius << std::endl <<
22:     std::endl;
23:     // Loop through body objects using the input file.
24:     for (int i = 0; i < number_planets; i++) {
25:         // Create a new object
26:         CelestialBody* tmp = new CelestialBody();
27:
28:         // Reads input into the object
29:         std::cin >> *tmp;
30:
31:         // Set the radius and the planet positions
32:         tmp->set_radius(universe_radius);
33:         tmp->set_position();
34:
35:         // Save the object to the vector
36:         body_vector.push_back(*tmp);
37:
38:         // Test the object in order to display output
39:         std::cout << *tmp;
40:     }
41:     // function call to draw universe
42:     drawUniverse();
43: }
44: void Universe::drawUniverse() {
45:     // SFML Window
46:     sf::RenderWindow window(sf::VideoMode(window_side, window_height),
47:     "Celestial Bodies"); // NOLINT
48:     // Change the framerate to make it easier to see the image moving.
49:     window.setFramerateLimit(1);
50:
51:     // Background image
52:     if (!background_image.loadFromFile("galaxy.jpg")) {
53:         return; // Quit if the file doesn't exist.
54:     }

```

```

55:
56: // Load the image into a texture
57:
58: background_texture.loadFromImage(background_image);
59:
60: // Load the texture into a sprite
61: background_sprite.setTexture(background_texture);
62:
63: // Set the position for the background image
64: background_sprite.setPosition(sf::Vector2f(0, 0));
65:
66: // Window loop
67: while (window.isOpen()) {
68:     // Process events
69:     sf::Event event;
70:
71:     while (window.pollEvent(event)) {
72:         // Close window : exit
73:         if (event.type == sf::Event::Closed) {
74:             window.close();
75:         } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
76:             window.close();
77:         }
78:     }
79:
80:     window.clear();
81:     // Draws the background
82:     window.draw(background_sprite);
83:
84:     // loops through the bodies till each body is displayed
85:     for (it = body_vector.begin(); it != body_vector.end(); it++) {
86:         window.draw(*it);
87:     }
88:
89: // displays the updated window
90:     window.display();
91: }
92: }

```

**NBody.h**

```

1: #ifndef NBODY_H // NOLINT
2:
3: #define NBODY_H // NOLINT
4:
5: #include <iostream>
6: #include <string>
7: #include <fstream>
8: #include <vector>
9: #include <SFML/System.hpp>
10: #include <SFML/Window.hpp>
11: #include <SFML/Graphics.hpp>
12:
13: // Constants for the window size.
14: const int window_height = 500;
15: const int window_side = 500;
16:
17: class CelestialBody: public sf::Drawable {
18: public:
19: // Constructor
20: CelestialBody();
21: // Parameterised Constructor
22: CelestialBody(float pos_x, float pos_y, float vel_x, float vel_y,
23: float obj_mass, float radius, std::string file_name);
24:
25: // sets the radius
26: void set_radius(float radius);
27:
28: // Sets the planets positions
29: void set_position();
30:
31: // Overridden operator >> for inputing from a file
32: friend std::istream& operator>> (std::istream &input, CelestialBody
33: &cBody);
34: // Overriddden operator << for displaying output from file
35: friend std::ostream& operator<< (std::ostream &output, CelestialBody
36: &cBody);
37: private:
38: // Draw method
39: void virtual draw(sf::RenderTarget& target, sf::RenderStates states)
                                                                    const;
40: // Member variables
41: float position_x, position_y;
42: float velocity_x, velocity_y;
43: float mass;
44: float c_radius;
45: std::string filename; // NOLINT
46: // Image related objects
47: sf::Image image;
48: sf::Sprite sprite;
49: sf::Texture texture;
50: };
51: class Universe {
52: // Create a vector of body objects
53: std::vector<CelestialBody>body_vector;

```

```
54:
55: // Display the vector of objects
56: std::vector<CelestialBody>::iterator it;
57: public:
58: void readInput();
59: void drawUniverse();
60: private:
61: float universe_radius;
62: int number_planets;
63: sf::Image background_image;
64: sf::Texture background_texture;
65: sf::Sprite background_sprite;
66: };
67: #endif // NOLINT
```



**NBody.cpp**

```

1: #include "NBody.h"
2:
3: // Default Constructor
4: CelestialBody::CelestialBody() {
5: // does nothing
6:     return;
7: }
8:
9: // Constructor with parameters
10:
11: CelestialBody::CelestialBody(float pos_x, float pos_y, float vel_x, float
12: vel_y, float obj_mass, float radius, std::string file_name) {
13: // Set member variables
14:     position_x = pos_x;
15:     position_y = pos_y;
16:     velocity_x = vel_x;
17:     velocity_y = vel_y;
18:     mass = obj_mass;
19:     filename = file_name;
20:
21:     // Load the image into an image object
22:     if (!image.loadFromFile(file_name)) {
23:         return; // Quit if the file doesn't exist.
24:     }
25:
26:     // Load the image into a texture
27:     texture.loadFromImage(image);
28:
29:     // Load the texture into a sprite
30:     sprite.setTexture(texture);
31:
32:     // Set the position from the Vector2f for position
33:     sprite.setPosition(sf::Vector2f(position_x, position_y));
34: }
35:
36: // Sets the universe radius
37: void CelestialBody::set_radius(float radius) {
38:     c_radius = radius;
39:     return;
40: }
41:
42: // Sets the planets position
43: void CelestialBody::set_position() {
44: double x = (position_x / c_radius) * (window_side / 2);
45: double y = (position_y / c_radius) * (window_height / 2);
46:
47: // Set the position for sprite
48: sprite.setPosition(x + (window_side/2), y + (window_height/2));
49: }
50: // Drawable method
51: void CelestialBody::draw(sf::RenderTarget& target,
52: sf::RenderStates states) const {
53: // tests the sprite to be drawn
54:     target.draw(sprite);

```

```

55: }
56:
57: // Overridden operator >> for inputing from a file
58: std::istream& operator>> (std::istream &input, CelestialBody &cBody) {
59: // Read input into the object
60:   input >> cBody.position_x >> cBody.position_y;
61:   input >> cBody.velocity_x >> cBody.velocity_y >> cBody.mass >>
62:   input >> cBody.filename;
63: // Load the image into an image object
64:   if (!cBody.image.loadFromFile(cBody.filename)) {
65:     return input;    // Quit if the file doesn't exist.
66:   }
67:
68:   // Load the image into a texture
69:   cBody.texture.loadFromImage(cBody.image);
70:
71:   // Load the texture into a sprite
72:   cBody.sprite.setTexture(cBody.texture);
73:
74:   // Set the initial position
75:   cBody.sprite.setPosition(sf::Vector2f(cBody.position_x,
76:   cBody.position_y));
77:   return input;
78: }
79:
80: // Overriddden operator <<
81:
82: std::ostream& operator<< (std::ostream &output, CelestialBody &cBody) {
83:   // For displaying all the data stored in the object.
84:   output << "Filename: " << cBody.filename << std::endl
85:   << "Position-x: " << cBody.position_x << std::endl
86:   << "Position-y: " << cBody.position_y << std::endl
87:   << "Velocity-x: " << cBody.velocity_x << std::endl
88:   << "Velocity-y: " << cBody.velocity_y << std::endl
89:   << "Body_Mass: " << cBody.mass << std::endl << std::endl;
90:
91:   return output;
92: }
93:
94:

```

## **PS2b: N-Body Simulation:**

### **About the Assignment**

This task is an extension of PS2a. The static universe from the previous assignment is used in PS2b, and I added simulation and animation to it using Newton's laws of universal gravitation and motion. The simulation is run using a method step that takes a time parameter (double seconds) and moves the CelestialBody object based on its internal velocity for that amount of time.

### **Key Concepts**

The primary concepts for this assignment are related to physics. They are as follows: -

- Newton's law of universal gravitation
- The superposition principle
- The second law of motion as stated by Newton

Apart from this I also used a few formulas to implement these concepts in PS2b, such as:

$$F = (G * M1 * M2) / R^2$$

$$R = \text{square root } (R2)$$

$$R2 = (\Delta x)^2 + (\Delta y)^2$$

$$\Delta x = x2 - x1$$

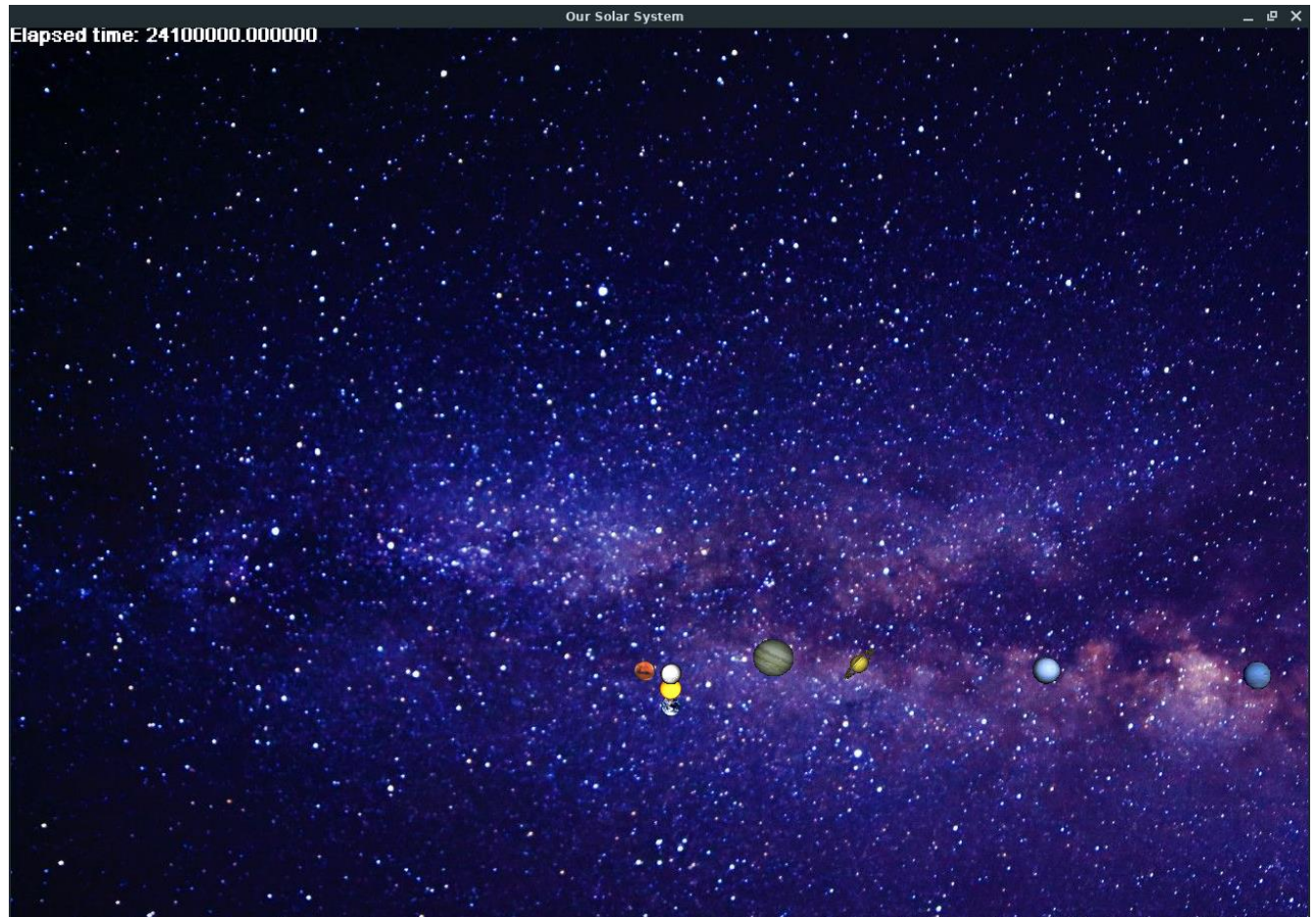
$$\Delta y = y2 - y1$$

*These formulas can be found in the find\_force\_x and find\_force\_y methods of the NBody.cpp*

I was able to simulate the movement of planets throughout the universe using these formulas. This was a difficult part of the assignment because getting the formulas right is critical to making the universe work properly.

### **What I Learned**

This assignment primarily taught me a practical application of physics as well as how to implement various equations in a program. It was difficult to get the equations correct because, as I discovered while programming this assignment, a single incorrect or slightly off equation can throw the entire solar system into chaos. When you've implemented them correctly, you'll get a nice simulation of the universe. I learned how to play music using SFML's audio library in addition to implementing the physics section. Although this was an extra credit assignment, it worked well with the body simulation.

**Screenshot:**

*Note: The above screenshot is just a capture of the running output. The sfml output has the bodies simulation with the background music.*

**Source code for PS2b NBody Simulation:****Makefile**

```
1: CXX= g++
2: CXXFLAGS= -Wall -Werror -std=c++0x -pedantic
3: LIBS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
4: LDFLAGS= $(LIBS)
5:
6: all: NBody
7:
8: NBody: main.o NBody.o
9:         $(CXX) main.o NBody.o -o NBody $(LDFLAGS)
10:
11: main.o: main.cpp NBody.h
12:         $(CXX) -c main.cpp NBody.h $(CXXFLAGS)
13:
14: NBody.o: NBody.cpp NBody.h
15:         $(CXX) -c NBody.cpp NBody.h $(CXXFLAGS)
16:
17: clean:
18:         rm -f main.o NBody.o
19:         rm -f NBody.h.gch
20:         rm -f NBody
21:
```

**Main.cpp**

```

1: #include "NBody.h"
2: double time_step;
3: double simu_time;
4: int main(int argc, char* argv[])
5: {
6:     if(argc != 3)
7:     {
8:         // ./NBody 157788000.0 25000.0 < planets.txt
9:         std::cout << "Usage: ./NBody [simulation time] [time step] <
                                planets.txt\n";
10:        return -1;
11:    }
12:
13:    // Get the simulation time / time step from the command line arguments
14:    std::string sim_time(argv[1]);
15:    std::string step_time(argv[2]);
16:
17:    // Debugging
18:    std::cout << "Simulation time: " << sim_time << "\n";
19:    std::cout << "Time Step: " << step_time << "\n\n";
20:
21:    simu_time = atof(sim_time.c_str());
22:    time_step = atof(step_time.c_str());
23:
24:    // Get the first two numbers in the text file.
25:    std::string num_planets, radius;
26:
27:    // Use cin to redirect the input
28:    std::cin >> num_planets;
29:    std::cin >> radius;
30:
31:    //Convert the input from std::strings to int / float
32:    int number_planets = std::atoi(num_planets.c_str());
33:    double universe_radius = std::atof(radius.c_str());
34:
35:
36:    // Displays the converted output
37:    std::cout << "Num of planets: " << number_planets << std::endl;
38:    std::cout << "Radius: " << universe_radius << std::endl << std::endl;
39:
40:    Universe universe;
41:
42:    universe.drawUniverse(number_planets,universe_radius);
43:
44:    return 0;
45:
46: }
47: void Universe::drawUniverse(int number_planets, double universe_radius)
48: {
49:
50:    double simulation_time = 0;
51:
52:    // Loop through, created body objects using the input file.
53:    for(int i = 0; i < number_planets; i++)

```

```

54:  {
55:
56:      std::shared_ptr<CelestialBody> tmp(new
57:          CelestialBody(universe_radius));
58:      std::cin >> *tmp;
59:
60:      body_vector.push_back(tmp);
61:
62:      std::cout << *tmp;
63:
64:      /*
65:      // Create a new object
66:      CelestialBody* tmp = new CelestialBody();
67:
68:      // Read input into the object
69:      std::cin >> *tmp;
70:
71:      // Set the radius and the planet positions.
72:      tmp->set_radius(universe_radius);
73:      tmp->set_position();
74:
75:      // Save the object to the vector
76:      body_vector.push_back(*tmp);
77:
78:      // Test the object (debugging)
79:      std::cout << *tmp;
80:      */
81:  }
82:
83:  // SFML Window
84:  sf::RenderWindow window(sf::VideoMode(window_side,
85:      window_height), "Our Solar System");
86:  // Change the framerate to make it easier to see the image moving.
87:  window.setFramerateLimit(60);
88:
89:  // Background image
90:  sf::Image background_image;
91:
92:  // Background image
93:  if (!background_image.loadFromFile("stars.jpg"))
94:  {
95:      std::cout<<"Image doesn't exist"<<std::endl;
96:      // Displayed if the file doesn't exist.
97:  }
98:  // Declare and load a font
99:  sf::Font time_font;
100:  time_font.loadFromFile("PIXEAB__.TTF");
101:
102:  // Text for displaying the current simulation time.
103:  sf::Text time_text;
104:
105:  // Select the font
106:  time_text.setFont(time_font); // font is a sf::Font
107:
108:  // Set the character size
109:  time_text.setCharacterSize(14); // in pixels, not points!
110:

```

```

111: // Set the color
112: time_text.setFillColor(sf::Color::White);
113:
114: // Load the music file
115: sf::Music music;
116: if(!music.openFromFile("music.ogg"))
117: {
118:     std::cout<<"Music file doesn't exist"<<std::endl;    // error
119: }
120:
121: // PLAY THE TUNE
122: music.play();
123: music.setLoop(true);
124:
125: // Load the image into a texture
126: sf::Texture background_texture;
127: background_texture.loadFromImage(background_image);
128:
129: // Load the texture into a sprite
130: sf::Sprite background_sprite;
131: background_sprite.setTexture(background_texture);
132:
133: while (window.isOpen())
134: {
135:     sf::Event event;
136:
137:     while(window.pollEvent(event))
138:     {
139:         if (event.type == sf::Event::Closed)
140:         {
141:             window.close();
142:         }
143:         else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
144:         {
145:             window.close();
146:         }
147:     }
148:
149:     window.clear();
150: // Draws the starry background (black backgrounds are so lame for a
151: solar system)
152:     window.draw(background_sprite);
153:
154: // Update the time string - I cast to an int to keep the time sane
155:     time_text.setString("Elapsed time: " +
                           std::to_string(simulation_time));
156:
157:     // Display the time in the left hand corner of the window
158:     window.draw(time_text);
159:
160:     // Calculate the net force on each body object
161:     x = body_vector.begin();
162:
163: // First loop goes through the whole body vector so we make sure each
164: // body object
165: // gets its net force updated.
166:     for(int a = 0; a < number_planets; a++)

```



```

167:     {
168:         y = body_vector.begin();
169:
170:         force_x = 0;
171:         force_y = 0;
172: // Second loop goes through the bodyvector again, so that the current
173: // body object
174: // gets effected by every other body object.
175:         for(int b = 0; b < number_planets; b++)
176:         {
177:             if(a != b) // Make sure not include the force on the body
178:             { // Basically - (earth, earth) shouldn't be a case.
179:                 force_x += find_force_x(*x, *y);
180:                 force_y += find_force_y(*x, *y);
181:             }
182:             y++;
183:         }
184:         // Update the forces inside the current object
185:         x->set_forces(force_x, force_y);
186:
187:         x++;
188:     }
189:
190:     // Display the vector of objects
191:     for(it = body_vector.begin(); it != body_vector.end(); it++)
192:     {
193:         window.draw(*it);
194:
195: // While we're displaying the objects, might as well move it one step!
196:         it->step(time_step);
197:
198: // Update image position
199:         it->set_position();
200:     }
201:
202:     window.display();
203:
204:     // Increase simulation time variable by the simulation step
205:     simulation_time += time_step;
206:
207:     // Stop when we've reached the simulation time
208:     if(simulation_time >= simu_time)
209:     {
210:         break;
211:     }
212: }
213: // For printing the final state of the universe
214: std::cout << "\n\n";
215: std::cout<<"Elapsed Time:
216: "<<std::to_string(simulation_time)<<std::endl;
217: for(it = body_vector.begin(); it != body_vector.end(); it++)
218: {
219:     std::cout << *it ;
220: }
221:
222:

```

**NBody.h**

```

1: #include <iostream>
2: #include <string>
3: #include <fstream>
4: #include <memory>
5: #include<vector>
6: #include<sstream>
7: #include<math.h>
8: #include <vector>
9: #include<SFML/Audio.hpp>
10: #include <SFML/System.hpp>
11: #include <SFML/Window.hpp>
12: #include <SFML/Graphics.hpp>
13:
14:
15: // Constants for the window size.
16: const int window_height = 1300;
17: const int window_side = 1300;
18:
19: // Physics Constants for gravity
20: const double gravity = 6.67e-11;
21:
22: class CelestialBody: public sf::Drawable
23: {
24: public:
25:
26: // Constructor
27: CelestialBody();
28:
29: CelestialBody(double);
30:
31: //Parameterised Constructor
32: CelestialBody(double pos_x, double pos_y,double vel_x, double vel_y,
33: double obj_mass, double radius, std::string file_name);
34:
35: //Mutator functions
36:
37: //sets the radius
38: void set_radius(float radius);
39:
40: // Sets the planets positions
41: void set_position();
42:
43: // Methods for calculating force components
44:
45: friend double find_force_x(CelestialBody &Body1,CelestialBody &Body2);
46:
47: friend double find_force_y(CelestialBody &Body1,CelestialBody &Body2);
48:
49: //Mutator function
50: void set_forces(double , double );
51:
52: //Step method
53: void step(double );
54:

```

```

55: // Overridden operator >> for inputing from a file
56: friend std::istream& operator>> (std::istream &input,CelestialBody
57:   &cBody);
58: // Overriddden operator << for displaying output from file
59: friend std::ostream& operator<< (std::ostream &output, CelestialBody
60:   &cBody);
61:
62: private:
63:
64: // Draw method
65: void virtual draw(sf::RenderTarget& target, sf::RenderStates states)
66: const;
67: public:
68:
69: // Member variables
70:
71: double position_x, position_y;
72: double velocity_x, velocity_y;
73: double acc_x, acc_y;
74: double F_x, F_y;
75: double mass;
76: double c_radius;
77: std::string filename;
78:
79: // Image related objects
80: sf::Image image;
81: sf::Sprite sprite;
82: sf::Texture texture;
83:
84: };
85:
86: class Universe
87: {
88:
89:
90: std::vector<std::shared_ptr <CelestialBody> > body_vector;
91:
92: /*
93: // Display the vector of objects
94: std::vector<std::shared_ptr <CelestialBody> >::iterator it;
95:
96: std::vector<std::shared_ptr <CelestialBody> >::iterator x,y;
97: // Create a vector of body objects
98:
99:
100:
101: std::vector<CelestialBody> body_vector;
102: */
103: // Display the vector of objects
104: std::vector<CelestialBody> ::iterator it;
105:
106: std::vector<CelestialBody> ::iterator x,y;
107:
108:
109: public:
110: void step(CelestialBody cBody,double);
111: void drawUniverse(int, double);

```

```
112:
113: private:
114: double force_x, force_y;
115: sf::Image background_image;
116: sf::Texture background_texture;
117: sf::Sprite background_sprite;
118:
119: };
```

**NBody.cpp**

```

1: #include "NBody.h"
2:
3: // Default Constructor
4: CelestialBody::CelestialBody()
5: {
6:     // Does nothing since I call the setter methods and the >> operator.
7:     return;
8: }
9: // Constructor with parameters
10: CelestialBody::CelestialBody(double pos_x, double pos_y, double vel_x,
11: double vel_y,
12: double obj_mass, double radius, std::string file_name)
13: {
14:     // Set member variables
15:     position_x = pos_x;
16:     position_y = pos_y;
17:     velocity_x = vel_x;
18:     velocity_y = vel_y;
19:     mass = obj_mass;
20:     filename = file_name;
21:
22:     // Load the image into an image object
23:     if (!image.loadFromFile(file_name))
24:     {
25:         return;    // Quit if the file doesn't exist.
26:     }
27:
28:     // Load the image into a texture
29:     texture.loadFromImage(image);
30:
31:     // Load the texture into a sprite
32:     sprite.setTexture(texture);
33:
34:     // Set the position from the Vector2f for position
35:     sprite.setPosition(sf::Vector2f(position_x, position_y));
36: }
37:
38:
39: // Sets the universe radius
40: void CelestialBody::set_radius(float radius)
41: {
42:     c_radius = radius;
43:     return;
44: }
45:
46:
47: // Sets the forces for a given object
48: void CelestialBody::set_forces(double forcex, double forcey)
49: {
50:     F_x = forcex;
51:     F_y= forcey;
52: }
53:
54:

```

```

55: // Finds the force (x) between two body objects
56: double find_force_x(CelestialBody &Body1, CelestialBody &Body2)
57: {
58:     /*
59:      * Formulas:
60:      *  $F = (G * M1 * M2) / R^2$ 
61:      *  $R = \sqrt{\text{delta\_x}^2 + \text{delta\_y}^2}$ 
62:      *  $R^2 = R \text{ squared}$ 
63:      *  $\text{delta\_x} = x2 - x1$ 
64:      *  $\text{delta\_y} = y2 - y1$ 
65:      */
66:     double dx = Body2.position_x - Body1.position_x;
67:     double dy = Body2.position_y - Body1.position_y;
68:     double R2 = pow(dx, 2) + pow(dy, 2);
69:     double R = sqrt(R2);
70:     double force = (gravity * Body1.mass * Body2.mass) / R2;
71:     double force_x = force * (dx / R);
72:
73:     return force_x;
74: }
75:
76:
77: // Finds the force (y) between two body objects
78: double find_force_y(CelestialBody &Body1, CelestialBody &Body2)
79: {
80:     /*
81:      * Formulas:
82:      *  $F = (G * M1 * M2) / R^2$ 
83:      *  $R = \sqrt{\text{delta\_x}^2 + \text{delta\_y}^2}$ 
84:      *  $R^2 = R \text{ squared}$ 
85:      *  $\text{delta\_x} = x2 - x1$ 
86:      *  $\text{delta\_y} = y2 - y1$ 
87:      */
88:     double dx = Body2.position_x - Body1.position_x;
89:     double dy = Body2.position_y - Body1.position_y;
90:     double R2 = pow(dx, 2) + pow(dy, 2);
91:     double R = sqrt(R2);
92:     double force = (gravity * Body1.mass * Body2.mass) / R2;
93:     double force_y = force * (dy / R);
94:
95:     return force_y;
96: }
97:
98:
99: void CelestialBody::step(double time_t)
100: {
101:     //F = ma;
102:     //acceleration = force/mass;
103:     acc_x = F_x / mass;
104:     acc_y = F_y / mass;
105:
106:     // v = v + at;
107:     velocity_x = velocity_x + (acc_x * time_t);
108:     velocity_y = velocity_y + (acc_y * time_t);
109:
110:     // d = d + vt;
111:     position_x = position_x + (velocity_x * time_t);

```

```

112:   position_y = position_y - (velocity_y * time_t);
113:
114: }
115:
116:
117: /*void Universe::step(CelestialBody cBody,double time_t)
118: {
119:
120:   cBody.acc_x = cBody.F_x / cBody.mass;
121:   cBody.acc_y = cBody.F_y / cBody.mass;
122:
123:   cBody.velocity_x = cBody.velocity_x + (cBody.acc_x * time_t);
124:   cBody.velocity_y = cBody.velocity_y - (cBody.acc_y * time_t);
125:
126:   cBody.position_x = cBody.position_x + (cBody.velocity_x * time_t);
127:   cBody.position_y = cBody.position_y - (cBody.velocity_y * time_t);
128:
129: }*/
130:
131: // Sets the planets position
132: void CelestialBody::set_position()
133: {
134:   double x = (position_x / c_radius) * (window_side / 2);
135:   double y = (position_y / c_radius) * (window_height / 2);
136:
137:   // Set the position for sprite
138:
139:   sprite.setPosition(x + (window_side/2), y + (window_height/2));
140:
141: }
142: // Drawable method
143: void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates
144: states) const
145: {
146:   // Testing outputting an image.
147:   target.draw(sprite);
148: }
149:
150:
151: // Overridden operator >> for inputing from a file
152: std::istream& operator>> (std::istream &input, CelestialBody &cBody)
153: {
154:   // Read input into the object
155:   input >> cBody.position_x >> cBody.position_y >> cBody.velocity_x >>
156:   cBody.velocity_y >> cBody.mass >> cBody.filename;
157:
158:   // Now set up the images
159:   // Just like the constructor
160:
161:   // Load the image into an image object
162:   if (!cBody.image.loadFromFile(cBody.filename))
163:   {
164:     return input;    // Quit if the file doesn't exist.
165:   }
166:
167:   // Load the image into a texture
168:   cBody.texture.loadFromImage(cBody.image);

```

```

169:
170: // Load the texture into a sprite
171: cBody.sprite.setTexture(cBody.texture);
172:
173: // Set the initial position
174: cBody.sprite.setPosition(sf::Vector2f(cBody.position_x,
175:   cBody.position_y));
176: // Set force / acceleration to 0.
177: cBody.F_x = 0;
178: cBody.F_y = 0;
179: cBody.acc_x = 0;
180: cBody.acc_y = 0;
181:
182: return input;
183: }
184:
185:
186: // Overriddden operator << for debugging
187: std::ostream& operator<< (std::ostream &output, CelestialBody &cBody)
188: {
189: // For debugging, output all the data stored in the object.
190:
191: output << std::scientific<<cBody.position_x << " " << std::scientific
192: <<cBody.position_y << " " << std::scientific <<cBody.velocity_x << " " <<
193: std::scientific <<cBody.velocity_y <<" " << cBody.mass <<" " <<
194: cBody.filename << std::endl;
195: return output;
196: }

```



## **PS3: Recursive Graphics**

### **About the Assignment**

We were tasked with implementing a variation of the Sierpinski triangle for this assignment. The assignment's main idea was to use recursion to create a complex-looking triangle. TFractal's main() function accepts two command-line arguments, L and N:

L, the length of the base equilateral triangle's side (double), and

N, the depth of the recursion (int)

Our program would then draw one triangle at depth 1, four triangles at depth 2, i.e. three child triangles drawn from each corner of the base triangle, and so on, recursively drawing triangles and their child triangles.

### **Key Concepts**

The concept of recursion was central to this program. I had to figure out a way to implement drawing the triangles recursively. While working on this assignment I found that using pointers to other triangles was the best way to do this – the first main triangle has three pointers,

```
Triangle* getLeft();
Triangle* getRight();
Triangle* getUnder();
```

and those have pointers to three more triangles, and so on until the maximum depth is reached.

I was able to recursively draw the triangles in the Triangle class by using pointers this can be illustrated using a simple tree diagram:

```

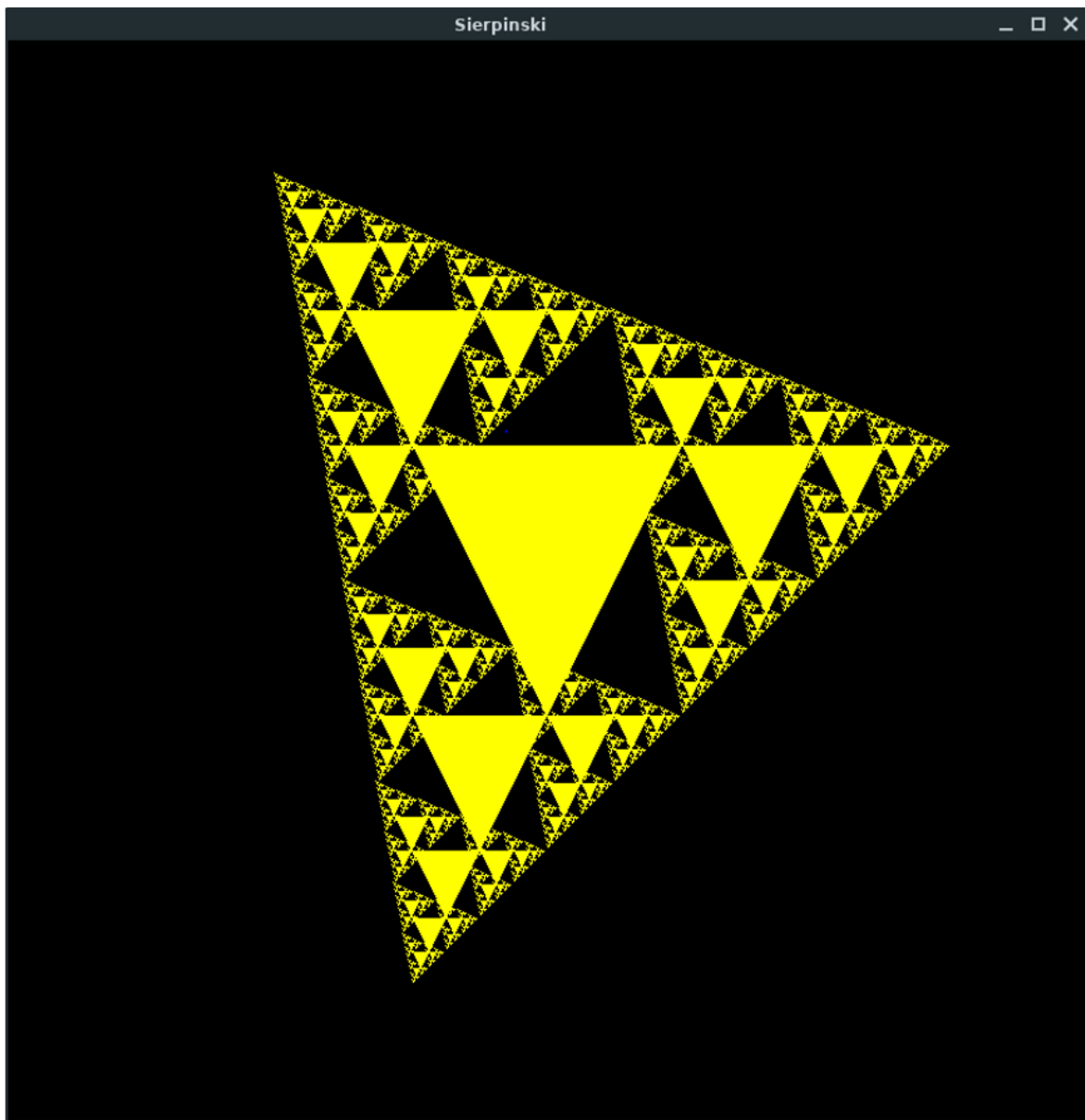
      Base   triangle
      /      |      \
    left  under  right
  /  |  \  /  |  \  /  |  \
pl pu pr pl pu pr pl pu pr
/|\|\/|\|\/|\|\/|\|\/|\|\/|\|
all the way to max depth.
```

This results in calling draw on each triangle, which then calls draw on all of their triangles, and so on until the recursion depth is reached.

## **What I Learned**

Working on this assignment helped me in understanding the concept of recursion more clearly. I have used recursion in many of my previous course assignment but never used it where I was able to use pointers to other objects in order to recursively go back and draw them out. This was one of the trickiest assignments because figuring out the co-ordinates from where the child triangles are going to be drawn from was a bit tough. But after that, the use of how to draw an object recursively was much easier. Apart from that I also added color to the triangle using the sfml color library which made my triangle look good. I also used cpplint to style check my cpp files, which taught me how to write proper cpp files. Cpplint was a fun tool to use. The majority of my classmates don't like using it, but I found it useful at times. It forced me to break some bad coding habits, such as writing extremely long lines of code(80+ characters).

Screenshot:



**Source code for PS3 Recursive Graphics:****Makefile**

```
1: CXX= g++
2: CXXFLAGS= -Wall -Werror -std=c++0x -pedantic
3: LIBS= -lsfml-graphics -lsfml-window -lsfml-system
4: LDFLAGS= $(LIBS)
5:
6: all: Triangle
7:
8: Triangle: TFractal.o Triangle.o
9:          $(CXX) TFractal.o Triangle.o -o Triangle $(LDFLAGS)
10:
11: TFractal.o: TFractal.cpp Triangle.h
12:           $(CXX) -c TFractal.cpp Triangle.h $(CXXFLAGS)
13:
14: Triangle.o: Triangle.cpp Triangle.h
15:           $(CXX) -c Triangle.cpp Triangle.h $(CXXFLAGS)
16:
17: clean:
18:       rm -f TFractal.o Triangle.o
19:       rm -f Triangle.h.gch
20:       rm -f Triangle
21:
```

**TFractal.cpp**

```

1: #include "Triangle.h"
2: int main(int argc, char* argv[]) {
3: if (argc != 3) {
4: std::cout << "Enter ./Triangle [Side] [Recursion_Depth] !!";
5: return-1;
6: }
7: std::string side(argv[1]);
8: std::string depth(argv[2]);
9:
10: double Length = std::atoi(side.c_str());
11: int width = static_cast<int> (Length) * 4;
12: int height = static_cast<int> (Length) * 4;
13: double Rec_depth = std::atoi(depth.c_str());
14: sf::RenderWindow window(sf::VideoMode(width, height), "Sierpinski");
15:
16: TFractal *obj = new TFractal();
17: obj->fTree(Rec_depth, Length, width, height);
18: Triangle* triangle5 = obj->base;
19: // Window loop
20: while (window.isOpen()) {
21: // Process events
22: sf::Event event;
23:
24: while (window.pollEvent(event)) {
25: // Close window : exit
26: if (event.type == sf::Event::Closed) {
27: window.close();
28: } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
29: window.close();
30: }
31: }
32: window.clear();
33: window.draw(*triangle5);
34: // Call the draw object in the sierpinkski class
35: window.display();
36: }
37: return 0;
38: }
39: void TFractal::fTree(int depth,
40: double side, int Window_Width, int Window_Height) {
41: base = new Triangle(side);
42: base->move((Window_Width/2) - (side/2), (Window_Height/2)-(side/2));
43: base->addLeftChild(fTree(base, depth));
44: base->addRightChild(fTree(base, depth));
45: base->addUnderChild(fTree(base, depth));
46: setChildPosition(base);
47: }
48: // void fTree(int depth);
49: Triangle* TFractal::fTree(Triangle* child, int depth) {
50: if(depth <= 0) {
51: return nullptr;
52: }
53: double baseLength = child->side;
54: // sf::Vector2f basePosition= triangle3->getPosition();

```

```

55: Triangle *triangleChild = new Triangle(baseLength/2);
56: triangleChild->addLeftChild(fTree(triangleChild, depth-1));
57: triangleChild->addRightChild(fTree(triangleChild, depth-1));
58: triangleChild->addUnderChild(fTree(triangleChild, depth-1));
59: return triangleChild;
60: }
61:
62: void TFractal::setChildPosition(Triangle* baseTriangle) {
63: if(baseTriangle == nullptr|| baseTriangle->triangle_Left ==nullptr) {
64: return;
65: }
66: sf::Vector2f basePosition{ baseTriangle->getPosition()};
67: double baseLength{ baseTriangle->side};
68: double childLength{ baseLength/2};
69: baseTriangle->triangle_Left->move(basePosition.x-childLength/2,
70:                                basePosition.y-childLength); // NOLINT
71: baseTriangle->triangle_Right->move(basePosition.x +
72:                                baseLength,basePosition.y);
73: baseTriangle->triangle_Under->move(basePosition.x,
74:                                basePosition.y+baseLength);
75:
76: setChildPosition(baseTriangle->triangle_Right);
77: setChildPosition(baseTriangle->triangle_Left);
78: setChildPosition(baseTriangle->triangle_Under);
79: }

```

**Triangle.h**

```

1: #ifndef Triangle_H // NOLINT
2: #define Triangle_H // // NOLINT
3:
4: #include <string.h>
5: #include <utility>
6: #include <iostream>
7: #include <SFML/System.hpp>
8: #include <SFML/Window.hpp>
9: #include <SFML/Graphics.hpp>
10: class Triangle : public sf::Drawable {
11: public:
12: Triangle(double);
13: double side;
14: void move(int x, int y);
15: sf::Vector2f getPosition();
16: void setOrigin(sf::Vector2f);
17: void addLeftChild(Triangle *base);
18: void addRightChild(Triangle *base);
19: void addUnderChild(Triangle *base);
20: Triangle* getLeft();
21: Triangle* getRight();
22: Triangle* getUnder();
23: Triangle *triangle_Left;
24: Triangle *triangle_Right;
25: Triangle *triangle_Under;
26: private:
27: void virtual draw(sf::RenderTarget& target, sf::RenderStates states)
                                const;
28: sf::ConvexShape base;
29: };
30: class TFractal {
31: public:
32: void fTree(int depth, double side, int Window_Height, int Window_Width);
33: // void fTree(int depth);
34: Triangle *base;
35: Triangle* fTree(Triangle* child, int depth);
36: void setChildPosition(Triangle *baseTriangle);
37: };
38: #endif // // NOLINT
39:

```

**Triangle.cpp**

```

1: #include "Triangle.h"
2:
3: Triangle::Triangle(double side) {
4:     base = sf::ConvexShape();
5:     base.setPointCount(3);
6:     base.setPoint(0, sf::Vector2f(0, 0));
7:     base.setPoint(1, sf::Vector2f(side, 0));
8:     base.setPoint(2, sf::Vector2f(side / 2, side));
9:     base.setFillColor(sf::Color::Yellow);
10:    this->side = side;
11:    return;
12: }
13: void Triangle::move(int x, int y) {
14:     base.move(sf::Vector2f(x, y));
15: }
16: sf::Vector2f Triangle::getPosition() {
17:     return base.getPosition();
18: }
19: void Triangle::setOrigin(sf::Vector2f origin) {
20:     this->base.setOrigin(origin);
21: }
22: // void Triangle::setFill(sf::Color color) {
23: //     triangle2.setFillColor(sf::Color::Blue); //sf::Color::Blue
24: // }
25: void Triangle::addLeftChild(Triangle *triangleLeft) {
26:     this->triangle_Left = triangleLeft;
27: }
28: void Triangle::addRightChild(Triangle *triangleRight) {
29:     this->triangle_Right = triangleRight;
30: }
31: void Triangle::addUnderChild(Triangle *triangleUnder) {
32:     this->triangle_Under = triangleUnder;
33: }
34: Triangle* Triangle::getLeft() {
35:     return triangle_Left;
36: }
37: Triangle* Triangle::getRight() {
38:     return triangle_Right;
39: }
40: Triangle* Triangle::getUnder() {
41:     return triangle_Under;
42: }
43: void Triangle::draw(sf::RenderTarget& target, sf::RenderStates states)
                                   const{
44:     target.draw(base);
45:     if (this->triangle_Under != nullptr) target.draw(*triangle_Under);
46:     if (this->triangle_Right != nullptr) target.draw(*triangle_Right);
47:     if (this->triangle_Left != nullptr) target.draw(*triangle_Left);
48:     // Testing outputting an image.
49: }

```



## **PS4a: Synthesizing a Plucked String Sound: CircularBuffer implementation with unit tests and exceptions**

### **About the Assignment**

I created a CircularBuffer for this assignment, with unit tests and exceptions. The main goal of the assignment was to implement the CircularBuffer, which stores values – in our case, 16 bit integers – by wrapping around like a circle array. Boost unit tests were also used to check the CircularBuffer for errors, and the CircularBuffer was designed to throw specific exceptions for specific errors. If you try to create a CircularBuffer with a capacity of 0 or less, you will get a `std::invalid_argument`, and if you try to enqueue, dequeue, or peek at an empty CircularBuffer, you will get a `std::runtime_error`.

### **Key Concepts**

The main concepts for this assignment focused on using exceptions to implement the CircularBuffer. To check for invalid actions, Try/Catch blocks were used. Boost was also used to validate the CircularBuffer, ensuring that it threw the correct exceptions when they were supposed to, and that no exceptions were thrown for valid actions. The `BOOST_REQUIRE_THROW` and `BOOST_REQUIRE_NO_THROW` determines the conditions that throws an exception and which do not.

### **What I Learned**

I learned a lot about exceptions and `cpplint` from this assignment. I had previously learned about exceptions in my Java class, but I had only used them a few times, and having not used them in a long time, I was a little rusty on the concept. It didn't take long, however, to figure them out. Working on this assignment helped me understand the Karplus-Strong algorithm concept and how to apply it in my program.



**Source code for PS4a CircularBuffer Implementation:****Makefile:**

```
1: CXX=g++
2: CXXFLAGS=-Wall -Werror -std=c++0x -pedantic
3: LDLIBS=-lboost_unit_test_framework
4: LDFLAGS=$(LDLIBS)
5:
6: OBJS= CircularBuffer.o test.o main.o
7:
8: all: ps4a main
9:
10: ps4a: test.o CircularBuffer.o
11:      $(CXX) test.o CircularBuffer.o -o ps4a $(LDFLAGS)
12:
13: main: main.o CircularBuffer.o
14:      $(CXX) main.o CircularBuffer.o -o main
15:
16: CircularBuffer.o: CircularBuffer.cpp CircularBuffer.h
17:      $(CXX) -c CircularBuffer.cpp CircularBuffer.h $(CXXFLAGS)
18:
19: test.o: test.cpp CircularBuffer.h
20:      $(CXX) -c test.cpp CircularBuffer.h $(CXXFLAGS)
21:
22: main.o: main.cpp CircularBuffer.h
23:      $(CXX) -c main.cpp CircularBuffer.h $(CXXFLAGS)
24:
25: clean:
26:      rm -f $(OBJS)
27:      rm -f CircularBuffer.h.gch
28:      rm -f ps4a main
```

**Main.cpp**

```

1: #include "CircularBuffer.h"
2:
3: int main() {
4:     std::cout << "Testing Main \n";
5:
6:     // Sets capacity of Buffer to 50
7:     CircularBuffer main_test(50);
8:
9:     // Inserts Element into the buffer
10:    main_test.enqueue(1);
11:    main_test.enqueue(2);
12:    main_test.enqueue(3);
13:
14:    // prints element at the front of buffer
15:    std::cout << "Peek: Item at the front is : " << main_test.peek() <<
16:    std::endl;
17:    // prints element deleted from the buffer
18:    std::cout << " Element Deleted : " << main_test.dequeue() << std::endl;
19:    std::cout << " Element Deleted : " << main_test.dequeue() << std::endl;
20:
21:    // Inserts Element into the buffer
22:    main_test.enqueue(4);
23:    main_test.enqueue(5);
24:
25:    // checks if buffer is empty or not
26:    if ( main_test.isEmpty() == true ) {
27:        std::cout << "Buffer is empty" << std::endl; // displayed if empty
28:    } else {
29:        std::cout << "Buffer is not empty" << std::endl; //displayed if not empty
30:    }
31:    if ( main_test.isFull() == true ) {
32:        std::cout << "Buffer is Full" << std::endl; // displayed if full
33:    } else {
34:        std::cout << "Buffer is not Full" << std::endl; // displayed if not full
35:    }
36:
37:    std::cout << "\n Listing Buffer items" << std::endl;
38:
39:    // displays the buffer
40:    main_test.output();
41:
42:    // second test for main
43:    CircularBuffer main_test2(3);
44:
45:    // inserts items into the buffer
46:    main_test2.enqueue(1);
47:    main_test2.enqueue(2);
48:    main_test2.enqueue(3);
49:
50:    // deletes items from the buffer returns item at front
51:    main_test2.dequeue();
52:    main_test2.dequeue();
53:    main_test2.dequeue();
54:

```

```
55: // inserts items into the buffer
56: main_test2.enqueue(4);
57: main_test2.enqueue(5);
58: main_test2.enqueue(6);
59:
60: // deletes item from the buffer
61: main_test2.dequeue();
62: main_test2.enqueue(7);
63: if ( main_test2.isEmpty() == true ) {
64:     std::cout << "Buffer is empty" << std::endl;
65: } else {
66:     std::cout << "Buffer is not empty" << std::endl;
67: }
68: if ( main_test2.isFull() == true ) {
69:     std::cout << "Buffer is Full" << std::endl;
70: } else {
71:     std::cout << "Buffer is not Full" << std::endl;
72: }
73:
74: std::cout << "Listing Buffer items" << std::endl;
75:
76: // displays the buffer
77: main_test2.output();
78:
79: return 0;
80: }
```

**CircularBuffer.h**

```

1:  #ifndef CIRCULARBUFFER_H // NOLINT
2:
3:  #define CIRCULARBUFFER_H // // NOLINT
4:
5:  #include <stdint.h>
6:  #include <iostream>
7:  #include <string>
8:  #include <sstream>
9:  #include <exception>
10: #include <stdexcept>
11: #include <vector>
12:
13: class CircularBuffer {
14: public:
15:     // API functions
16:
17:     // Empty ring buffer, with given max capacity.
18:     explicit CircularBuffer(int capacity);
19:     int size(); // return # of items in the buffer.
20:     bool isEmpty(); // is size == 0?
21:     bool isFull(); // is size == capacity?
22:     void enqueue(int16_t x); // add item x to the end.
23:     int16_t dequeue(); // delete and return item from the front
24:     int16_t peek(); // return (don't delete) item from front
25:
26:     // Other functions
27:     void output();
28:
29: private:
30:     std::vector<int16_t> buffer;
31:     int first_element;
32:     int last_element;
33:     int buff_capacity;
34:     int buff_size;
35: };
36: #endif // // NOLINT

```

**CircularBuffer.cpp**

```

1: #include "CircularBuffer.h"
2:
3: // Creates an empty Circular buffer, with given capacity.
4: CircularBuffer::CircularBuffer(int capacity) {
5:     if (capacity < 1) {
6:         throw
7:             std::invalid_argument("Constructor capacity must be greater than 0");
8:     }
9:
10:    last_element = 0;
11:    first_element = 0;
12:    buff_size = 0;
13:    buff_capacity = capacity;
14:    buffer.resize(capacity);
15:
16:    return;
17: }
18:
19:
20: // Return # of items in the buffer.
21: int CircularBuffer::size() {
22:     return buff_size;
23: }
24:
25:
26: // Is size == 0?
27: bool CircularBuffer::isEmpty() {
28:     // Determine if the CircularBuffer is empty.
29:     if (buff_size == 0) {
30:         return true;
31:     } else {
32:         return false;
33:     }
34: }
35:
36:
37: // Is size == capacity?
38: bool CircularBuffer::isFull() {
39:     // Determine if size equals capacity.
40:     if (buff_size == buff_capacity) {
41:         return true;
42:     } else {
43:         return false;
44:     }
45: }
46:
47:
48: // Add item x to the end.
49: void CircularBuffer::enqueue(int16_t x) {
50:     // See if the buffer is full
51:     if (isFull()) {
52:         throw
53:             std::runtime_error("enqueue: can't enqueue to a full ring");
54:     }

```

```

55:
56: // Check to see if we need to loop last back around to 0.
57: if (last_element >= buff_capacity) {
58:     last_element = 0;
59: }
60:
61: // If we don't throw any exceptions, then continue on!
62: buffer.at(last_element) = x;
63:
64: // Increase counter variables.
65: last_element++;
66: buff_size++;
67: }
68:
69:
70: // Delete and return item from the front
71: int16_t CircularBuffer::dequeue() {
72:     if (isEmpty()) {
73:         throw
74:             std::runtime_error("dequeue: can't dequeue to an empty ring");
75:     }
76:
77:     // Remove from the front.
78:     int16_t first = buffer.at(first_element);
79:     buffer.at(first_element) = 0;
80:
81:     // Decrease counter variables.
82:     first_element++;
83:     buff_size--;
84:
85:     // Check to see if we need to loop first back around to 0.
86:     if (first_element >= buff_capacity) {
87:         first_element = 0;
88:     }
89:
90:     return first;
91: }
92:
93:
94: // Return (don't delete) item from the front.
95: int16_t CircularBuffer::peek() {
96:     // This is an easy function - return the first buffer position.
97:     if (isEmpty()) {
98:         throw
99:             std::runtime_error("peek: can't peek an empty ring");
100:     }
101:
102:     return buffer.at(first_element);
103: }
104:
105: // Dumps the variables to stdout
106: void CircularBuffer::output() {
107:     std::cout << " First item in the Buffer: " << first_element <<
108:                                     std::endl;
109:     std::cout << " Last item in the Buffer: " << last_element <<
110:                                     std::endl;
111:     std::cout << " Capacity of the Buffer: " << buff_capacity <<

```



```

                                std::endl;
110:
111:   std::cout << " Buffer Size: " << buff_size << std::endl;
112:   std::cout << " Vector size: " << buffer.size() << std::endl;
113:   std::cout << " Vector capacity: " << buffer.capacity() << std::endl;
114:   std::cout << " Display Buffer (with no blanks): " << std::endl;
115:
116:   int x = 0;
117:   int y = first_element;
118:
119:   while (x < buff_size) {
120:       // Make the loop go back to 0 to continue printing.
121:       if (y >= buff_capacity) {
122:           y = 0;
123:       }
124:
125:       std::cout << buffer[y] << " ";
126:       y++;
127:       x++;
128:   }
129:
130:   std::cout << "\n Dumps the entire buffer (including blanks): \n";
131:
132:   for (int i = 0; i < buff_capacity; i++) {
133:       std::cout << buffer[i] << " ";
134:   }
135:
136:   std::cout << "\n\n";
137: }

```

**Test.cpp**

```

1: #define BOOST_TEST_DYN_LINK
2: #define BOOST_TEST_MODULE Main
3: #include <boost/test/unit_test.hpp>
4:
5: #include "CircularBuffer.h"
6:
7: // CIRCULAR_BUFFER is the test suite name
8: BOOST_AUTO_TEST_SUITE(CIRCULAR_BUFFER)
9:
10: // Tests various aspects of the constructor.
11: BOOST_AUTO_TEST_CASE(Constructor) {
12:     // constructor test - shouldn't fail for capacity greater than 0.
13:     BOOST_REQUIRE_NO_THROW(CircularBuffer(10));
14:
15:     // BOOST_REQUIRE_NO_THROW(CircularBuffer(0));
16:
17:     // BOOST_REQUIRE_NO_THROW(CircularBuffer(-1));
18:
19:     // These should fail when capacity value is not equal to 0 and -1
20:     BOOST_REQUIRE_THROW(CircularBuffer(0), std::exception);
21:     BOOST_REQUIRE_THROW(CircularBuffer(0), std::invalid_argument);
22:     BOOST_REQUIRE_THROW(CircularBuffer(-1), std::invalid_argument);
23: }
24:
25:
26: // Checks the size() method
27: BOOST_AUTO_TEST_CASE(Size) {
28:     CircularBuffer test1(1);
29:
30:     // This should be size 0.
31:     BOOST_REQUIRE(test1.size() == 0);
32:
33:     test1.enqueue(5);
34:
35:     // This should be size 1.
36:     BOOST_REQUIRE(test1.size() == 1);
37:
38:     test1.dequeue();
39:     BOOST_REQUIRE(test1.size() == 0);
40: }
41: // Checks the isEmpty() method
42: BOOST_AUTO_TEST_CASE(isEmpty) {
43:     // This should be true
44:     CircularBuffer test2_a(5);
45:     BOOST_REQUIRE(test2_a.isEmpty() == true);
46:
47:     // This should be false
48:     CircularBuffer test2_b(5);
49:     test2_b.enqueue(5);
50:     BOOST_REQUIRE(test2_b.isEmpty() == false);
51: }
52:
53:
54: // Checks the isFull() method

```

```

55: BOOST_AUTO_TEST_CASE(isFull) {
56:     CircularBuffer test3_a(5);
57:     BOOST_REQUIRE(test3_a.isFull() == false);
58:
59:     CircularBuffer test3_b(1);
60:     test3_b.enqueue(5);
61:     BOOST_REQUIRE(test3_b.isFull() == true);
62: }
63:
64:
65: // Test enqueue
66: BOOST_AUTO_TEST_CASE(Enqueue) {
67:     // These test basic enqueueing
68:     CircularBuffer test4(5);
69:
70:     BOOST_REQUIRE_NO_THROW(test4.enqueue(1));
71:     BOOST_REQUIRE_NO_THROW(test4.enqueue(2));
72:     BOOST_REQUIRE_NO_THROW(test4.enqueue(3));
73:     BOOST_REQUIRE_NO_THROW(test4.enqueue(4));
74:     BOOST_REQUIRE_NO_THROW(test4.enqueue(5));
75:     BOOST_REQUIRE_THROW(test4.enqueue(6), std::runtime_error);
76: }
77:
78:
79: // Test dequeue
80: BOOST_AUTO_TEST_CASE(Dequeue) {
81:     CircularBuffer test5(5);
82:
83:     test5.enqueue(0);
84:     test5.enqueue(1);
85:     test5.enqueue(2);
86:
87:     BOOST_REQUIRE(test5.dequeue() == 0);
88:     BOOST_REQUIRE(test5.dequeue() == 1);
89:     BOOST_REQUIRE(test5.dequeue() == 2);
90:     BOOST_REQUIRE_THROW(test5.dequeue(), std::runtime_error);
91: }
92:
93: BOOST_AUTO_TEST_SUITE_END()

```

## **PS4b: StringSound implementation and SFML audio output**

### **About the Assignment**

In the second part of PS4, I used the same CircularBuffer created in PS4a to implement the Karplus-Strong guitar string simulation, and generate a stream of string samples for audio playback under keyboard control. The main task was to use the methods in StringSound to simulate guitar playing, such as plucking, ticing, sampling, and so on. Finally, we made the main program, KSGuitarSim, that has all the 37 notes to which the sound is played. Depending on the key pressed on the Keyboard a different sound is generated.

### **Key Concepts**

The Karplus-Strong algorithm was used as the main algorithm in this assignment to simulate guitar plucking. The Karplus-Strong algorithm works by modeling frequencies, by taking the first two values, averaging them, and then multiplying the result by the energy decay factor, which in our case was 0.996. This, in conjunction with the CircularBuffer, allowed us to model sound (to a degree) and make it appear as if a guitar string was being plucked. Boost was also used to check if the StringSound was implemented correctly, also ensuring that it threw the correct exceptions when they were supposed to, and that no exceptions were thrown for valid actions. The BOOST\_REQUIRE\_THROW and BOOST\_REQUIRE\_NO\_THROW determines the conditions that throws an exception and which do not.

### **What I Learned**

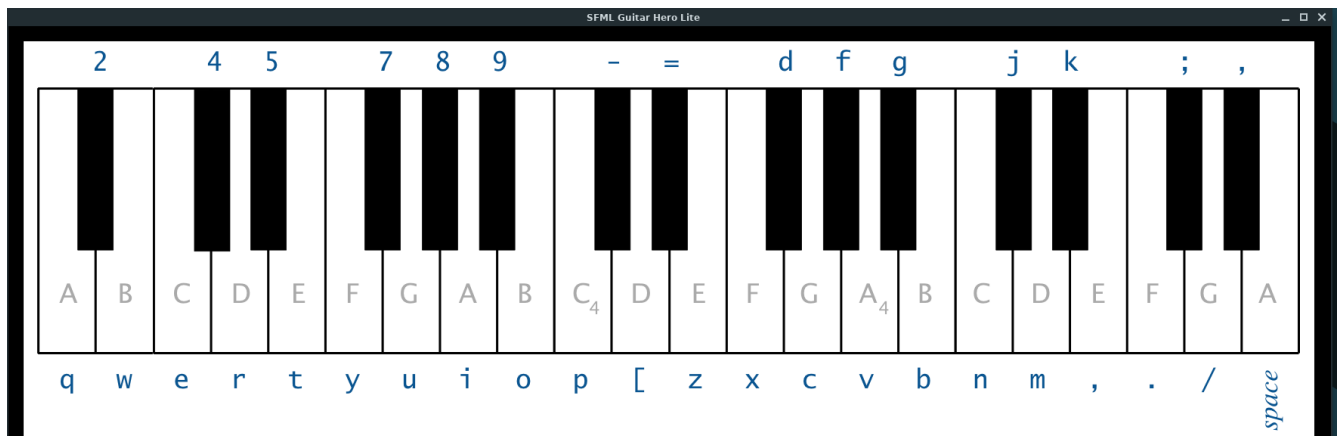
The Karplus-Strong algorithm was fascinating to study, and it taught me how to model sound in a program. It was also difficult to get the Karplus-Strong update to work properly, and I experienced a lot of segmentation faults when I first wrote the program because I was using pointers. But I figured out how to make them work again. I also learned how to use the std::mt19937 function, which I used to generate a seed here. I also learned how to use lambda expressions in my program. Aside from that, SFML's Keyboard library is very useful for controlling a piano, guitar, and so on.

**Screenshot:**

```

Linux Lite Terminal -
File Edit View Terminal Tabs Help
osboxes ~ > ps4b make all
g++ -c KSGuitarSim.cpp StringSound.h -Wall -Werror -std=c++0x -pedantic
g++ -c StringSound.cpp StringSound.h -Wall -Werror -std=c++0x -pedantic
g++ -c CircularBuffer.cpp CircularBuffer.h -Wall -Werror -std=c++0x -pedantic
g++ KSGuitarSim.o StringSound.o CircularBuffer.o -o KSGuitarSim -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
g++ -c Guitartest.cpp -lboost_unit_test_framework
g++ Guitartest.o StringSound.o CircularBuffer.o -o Guitartest -lboost_unit_test_framework
osboxes ~ > ps4b ./Guitartest
Running 2 test cases...
Sample is: 0
Sample is: 2000
Sample is: 4000
Sample is: -10000
*** No errors detected
osboxes ~ > ps4b ./KSGuitarSim
Setting vertical sync not supported

```



**Note:** On every keyboard stroke for the specified characters in the above image the Guitar tone is played.

## **Source code for PS4b StringSound implementation and SFML audio output:**

### **Makefile**

```

1: CXX=g++
2: CXXFLAGS=-Wall -Werror -std=c++0x -pedantic
3: LDLIBS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
4: Boost=-lboost_unit_test_framework
5: LDFLAGS=$(LDLIBS)
6:
7: OBJS= CircularBuffer.o StringSound.o KSGuitarSim.o Guitartest.o
8: EXES= KSGuitarSim Guitartest
9:
10: all: KSGuitarSim Guitartest
11:
12: # PS5B executable
13: KSGuitarSim: KSGuitarSim.o StringSound.o CircularBuffer.o
14:             $(CXX) KSGuitarSim.o StringSound.o CircularBuffer.o -o
15:                 KSGuitarSim $(LDFLAGS)
16: # GStest executable
17: Guitartest: Guitartest.o StringSound.o CircularBuffer.o
18:             $(CXX) Guitartest.o StringSound.o CircularBuffer.o -o Guitartest
19:                 $(Boost)
20: # Object files
21: KSGuitarSim.o: KSGuitarSim.cpp StringSound.h
22:             $(CXX) -c KSGuitarSim.cpp StringSound.h $(CXXFLAGS)
23:
24: StringSound.o: StringSound.cpp StringSound.h
25:             $(CXX) -c StringSound.cpp StringSound.h $(CXXFLAGS)
26:
27: CircularBuffer.o: CircularBuffer.cpp CircularBuffer.h
28:             $(CXX) -c CircularBuffer.cpp CircularBuffer.h $(CXXFLAGS)
29:
30: Guitartest.o: Guitartest.cpp
31:             $(CXX) -c Guitartest.cpp $(Boost)
32:
33: # Cleanup object files
34: clean:
35:         rm -f $(OBJS) $(EXES)
36:         rm *.gch
37:

```

**KSGuitarSim.cpp**

```

1: #include <math.h>
2: #include <limits.h>
3: #include <iostream>
4: #include <string>
5: #include <exception>
6: #include <stdexcept>
7: #include <vector>
8: #include <SFML/Graphics.hpp>
9: #include <SFML/System.hpp>
10: #include <SFML/Audio.hpp>
11: #include <SFML/Window.hpp>
12:
13: #include "StringSound.h"
14: #include "CircularBuffer.h"
15:
16: #define CONCERT_A 220.0
17: #define SAMPLES_PER_SEC 44100
18:
19:
20: vector<sf::Int16> makeSamplesFromString(StringSound gs) {
21:     std::vector<sf::Int16> samples;
22:
23:     gs.pluck();
24:     int duration = 8; // seconds
25:     int i;
26:     for (i = 0; i < SAMPLES_PER_SEC * duration; i++) {
27:         gs.tic();
28:         samples.push_back(gs.sample());
29:     }
30:
31:     return samples;
32: }
33: int main() {
34:     sf::RenderWindow window(sf::VideoMode(1750, 550),
35: "SFML Guitar Hero Lite");
36:     sf::Event event;
37:     double freq;
38:     std::vector<StringSound*> gs_list;
39:     std::vector<sf::SoundBuffer*> buffers;
40:     std::vector<sf::Sound*> sounds;
41:
42:
43:     // we're reusing the freq and samples vars, but
44:     // there are separate copies of StringSound, SoundBuffer, and Sound
45:     // for each note
46:     //
47:     // StringSound is based on freq
48:     // samples are generated from StringSound
49:     // SoundBuffer is loaded from samples
50:     // Sound is set to SoundBuffer
51:
52:     vector<sf::Int16> samples;
53:     for (int i = 0; i < 37; i++) {
54: // lamda expression

```

```

55:         auto fun_freq = [&] {
56:     return CONCERT_A * pow(2, (i - 24.0) / 12.0);
57: };
58: /* freq = CONCERT_A * pow(2, (i - 24.0) / 12.0); */
59:     freq = fun_freq();
60:
61:     gs_list.push_back(new StringSound(freq));
62:     sounds.push_back(new sf::Sound());
63:     buffers.push_back(new sf::SoundBuffer());
64:     auto *sound = sounds[i];
65:     auto *buf = buffers[i];
66:     auto& gs = *gs_list[i];
67:     samples = makeSamplesFromString(gs);
68:     if (!buf->loadFromSamples(&samples[0], samples.size(), 2,
                                SAMPLES_PER_SEC))
69:         throw std::runtime_error("sf::SoundBuffer: failed to load from
                                samples.");
70:     sound->setBuffer(*buf);
71: }
72: std::vector<int> keyboard = {sf::Keyboard::Q, sf::Keyboard::Num2,
sf::Keyboard::W, sf::Keyboard::E, sf::Keyboard::Num4, // NOLINT
73: sf::Keyboard::R, sf::Keyboard::Num5, sf::Keyboard::T, sf::Keyboard::Y,
sf::Keyboard::Num7, // NOLINT
74: sf::Keyboard::U, sf::Keyboard::Num8, sf::Keyboard::I,
sf::Keyboard::Num9, sf::Keyboard::O, // NOLINT
75: sf::Keyboard::P, sf::Keyboard::Dash, sf::Keyboard::LBracket,
sf::Keyboard::Equal, // NOLINT
76: sf::Keyboard::Z, sf::Keyboard::X, sf::Keyboard::D, sf::Keyboard::C,
sf::Keyboard::F, // NOLINT
77: sf::Keyboard::V, sf::Keyboard::G, sf::Keyboard::B, sf::Keyboard::N,
sf::Keyboard::J, // NOLINT
78: sf::Keyboard::M, sf::Keyboard::K, sf::Keyboard::Comma,
sf::Keyboard::Period, // NOLINT
79: sf::Keyboard::SemiColon,
80: sf::Keyboard::Slash, sf::Keyboard::Quote, sf::Keyboard::Space};
81: // Load a sprite to display
82: sf::Texture texture;
83:
84:     if (!texture.loadFromFile("keyboard.png"))
85:         // exits the code giving an error message if image doesn't exist
86:         return EXIT_FAILURE;
87:
88:     // Create SFML sprite
89:     sf::Sprite sprite(texture);
90:
91:     // sets the position of the sprite
92:     sprite.setPosition(sf::Vector2f(20, 20));
93:
94:     while (window.isOpen()) {
95:         while (window.pollEvent(event)) {
96:             if (event.type == sf::Event::Closed)
97:                 window.close();
98:             if (event.type == sf::Event::KeyPressed) {
99:                 for (unsigned long i = 0; i < keyboard.size(); i++) { // NOLINT
100:                     if (event.key.code == keyboard[i]) {
101:                         /* std::cout << "Keyboard key is: " << keyboard[i]<< "\n";
102:                         std::cout << "Attempting to play sound...\n"; */

```



```

103:                 sounds[i]->play();
104:                 break;
105:             }
106:         }
107:     } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
108:         window.close();
109:     }
110: }
111: window.clear();
112: window.draw(sprite);
113: window.display();
114: }
115: for (ulong i = 0; i < gs_list.size(); i++) {
116:     delete gs_list[i];
117:     delete sounds[i];
118:     // delete buffers[i];
119: }
120: /* gs_list.clear();
121:    sounds.clear();*/
122:    return 0;
123: }

```

**StringSound.h**

```

1: #ifndef STRINGSOUND_H // NOLINT
2: #define STRINGSOUND_H // NOLINT
3:
4: #include <time.h>
5: #include <vector>
6: #include <string>
7: #include <random>
8: #include <chrono> // NOLINT
9: #include <SFML/System.hpp>
10: #include "CircularBuffer.h"
11:
12: using std::vector;
13:
14: class StringSound {
15: private:
16: CircularBuffer *cb{};
17: int length;
18: int _time{};
19: std::mt19937 *cd;
20:
21: public:
22: // create a guitar string of the given freq using a rate of 44,100
23: explicit StringSound(double frequency);
24: // create a guitar string with size and initial values of the vector init
25: explicit StringSound(std::vector<sf::Int16> init);
26: // pluck the guitar string by replacing the buffer with random values
27: void pluck();
28: // advance the simulation one time step
29: void tic();
30: // return the current sample
31: sf::Int16 sample();
32: // return number of times tic was called so far
33: int time();
34: ~StringSound();
35: };
36:
37: #endif // NOLINT

```

**StringSound.cpp**

```

1: #include <chrono>
2: #include "StringSound.h"
3:
4: StringSound::StringSound(double frequency) {
5:     if ( frequency <= 0 ) {
6:         throw
7:         std::invalid_argument("Constructor: frequency can't be a zero or neg");
8:     }
9:     length = ceil(44100 / frequency);
10:    cb = new CircularBuffer(length);
11:    for ( int i = 0; i < length; i++ ) {
12:        cb->enqueue((int16_t)0);
13:    }
14:    unsigned seed =
std::chrono::system_clock::now().time_since_epoch().count();
15:    cd = new std::mt19937(seed);
16: }
17:
18: void StringSound::pluck() {
19:     for ( int i = 0; i < length; i++ ) {
20:         cb->dequeue();
21:     }
22:     /* for ( int i = 0; i < length ; i++)
23:         // fill CircularBuffer with random values ranging from -32768 to 32767.
24:         cb->enqueue((sf::Int16)(rand() & 0xffff)); // NOLINT
25:     */
26:     for (int i = 0; i < length; i++)
27:         cb->enqueue((*cd)());
28: }
29:
30: StringSound::StringSound(std::vector<sf::Int16> init) {
31:     cb = new CircularBuffer(init.size());
32:     for (int n : init) {
33:         cb->enqueue(n);
34:     }
35:     unsigned seed =
std::chrono::system_clock::now().time_since_epoch().count();
36:     cd = new std::mt19937(seed);
37: }
38:
39: void StringSound::tic() {
40:     _time++;
41:     /*int16_t t = cb->dequeue();
42:     cb->enqueue((int16_t) ((t + cb->peek()) / 2.0 * 0.996));
43:     */
44:     int16_t t = cb->dequeue();
45:     int16_t u = cb->peek();
46:     int16_t avg = [&] { return (t + u) / 2; } ();
47:     cb->enqueue(avg*0.996);
48: }
49:
50: sf::Int16 StringSound::sample() {
51:     return cb->peek();
52: }

```

```
53:
54: int StringSound::time() {
55: return _time;
56: }
57:
58: StringSound::~~StringSound() {
59: delete cb;
60: }
```

**CircularBuffer.h**

```

1: #ifndef CIRCULARBUFFER_H // NOLINT
2: #define CIRCULARBUFFER_H // NOLINT
3:
4: #include <stdint.h>
5: #include <iostream>
6: #include <string>
7: #include <sstream>
8: #include <exception>
9: #include <stdexcept>
10: #include <vector>
11:
12: class CircularBuffer {
13: private:
14:     int16_t *data;
15:     int count;
16:     int capacity;
17:     int front;
18:     int end;
19:
20: public:
21:     // Empty Circular buffer, with given max capacity.
22:     explicit CircularBuffer(int capacity);
23:     ~CircularBuffer();
24:     // return # of items in the buffer
25:     int size();
26:     // is size == 0?
27:     bool isEmpty();
28:     // is size == capacity?
29:     bool isFull();
30:     // add item x to the end
31:     void enqueue(int16_t x);
32:     // delete and return item from the front
33:     int16_t dequeue();
34:     // return (don't delete) item from the front
35:     int16_t peek();
36: };
37:
38: #endif // NOLINT

```

**CircularBuffer.cpp**

```

1: #include <stdexcept>
2: #include "CircularBuffer.h"
3:
4: CircularBuffer::CircularBuffer(int capacity):capacity(capacity) {
5: if (capacity < 1) {
6: throw
7: std::invalid_argument("CB constructor: capacity must be greater than 0");
8: }
9: data = new int16_t[capacity];
10: front = 0;
11: end = -1;
12: count = 0;
13: }
14:
15: int CircularBuffer::size() {
16: return count;
17: }
18:
19: bool CircularBuffer::isEmpty() {
20: if ( count == 0 ) {
21: return true;
22: } else {
23: return false;
24: }
25: }
26:
27: bool CircularBuffer::isFull() {
28: if ( count == capacity ) {
29: return true;
30: } else {
31: return false;
32: }
33: }
34:
35: void CircularBuffer::enqueue(int16_t x) {
36: if ( isFull() ) {
37: throw std::runtime_error("enqueue: can't enqueue to a full ring");
38: }
39: end = (end+1) % capacity;
40: count++;
41: data[end] = x;
42: }
43:
44: int16_t CircularBuffer::dequeue() {
45: if (isEmpty()) {
46: throw
47: std::runtime_error("dequeue: can't dequeue to an empty ring");
48: }
49: int16_t v = peek();
50: front = (front+1) % capacity;
51: count--;
52: return v;
53: }
54:

```

```
55: int16_t CircularBuffer::peek() {
56: if (isEmpty()) {
57: throw
58: std::runtime_error("enqueue: can't peek or dequeue from a empty ring");
59: }
60: return data[front];
61: }
62:
63: CircularBuffer::~CircularBuffer() {
64: delete[] data;
65: }
66:
67:
```

**Guitartest.cpp**

```

1: #define BOOST_TEST_DYN_LINK
2: #define BOOST_TEST_MODULE Main
3:
4: #include <vector>
5: #include <exception>
6: #include <stdexcept>
7: #include <boost/test/unit_test.hpp>
8:
9: #include "StringSound.h"
10:
11: // CIRCULAR_BUFFER is the test suite name
12: BOOST_AUTO_TEST_SUITE (STRING_SOUND)
13:
14: BOOST_AUTO_TEST_CASE (Constructor) {
15: BOOST_REQUIRE_NO_THROW (StringSound(10));
16:
17: BOOST_REQUIRE_THROW (StringSound(0), std::exception);
18: BOOST_REQUIRE_THROW (StringSound(0), std::invalid_argument);
19: BOOST_REQUIRE_THROW (StringSound(-1), std::invalid_argument);
20: }
21: BOOST_AUTO_TEST_CASE (StringSound_test) {
22: std::vector<sf::Int16> v;
23:
24: v.push_back(0);
25: v.push_back(2000);
26: v.push_back(4000);
27: v.push_back(-10000);
28:
29: BOOST_REQUIRE_NO_THROW (StringSound gs = StringSound(v));
30:
31: StringSound gs = StringSound(v);
32:
33: // GS is 0 2000 4000 -10000
34: std::cout << "Sample is: " << gs.sample() << "\n";
35: BOOST_REQUIRE (gs.sample() == 0);
36:
37: gs.tic();
38: // it's now 2000 4000 -10000 996
39: std::cout << "Sample is: " << gs.sample() << "\n";
40: BOOST_REQUIRE (gs.sample() == 2000);
41:
42: gs.tic();
43: // it's now 4000 -10000 996 2988
44: std::cout << "Sample is: " << gs.sample() << "\n";
45: BOOST_REQUIRE (gs.sample() == 4000);
46:

```



```
47: gs.tic();
48: // it's now -10000 996 2988 -2988
49: std::cout << "Sample is: " << gs.sample() << "\n";
50: BOOST_REQUIRE(gs.sample() == -10000);
51: }
52:
53: BOOST_AUTO_TEST_SUITE_END()
```

## **PS5: DNA Sequence Alignment**

### **About the Assignment**

In this project we were asked to write a program to compute the optimal sequence alignment of two DNA strings using a powerful algorithmic design paradigm known as dynamic programming. Also using Valgrind, a memory analysis tool to measure and report the space and time performance of the implementation. A key idea for this program was also to use dynamic programming to make calculating the edit distance efficient.

### **Key Concepts**

The key concept that was used in this project was the Needleman-Wunsch method, which is a method of using dynamic programming to calculate subproblems and then using those subproblems to find the main solution. The key idea of dynamic programming is to break up a large computational problem into smaller subproblems, store the answers to those smaller subproblems, and, eventually, use the stored answers to solve the original problem. For this, I used a  $M \times N$  matrix and first computed the edit distance between two strings. And then using those solutions to find the next round of edit distances, until you recover the optimal alignment itself. To determine the choice that led to  $\text{opt}[i][j]$ . We accomplished this by following a few simple rules:

**Case 1.** The optimal alignment matches  $x[i]$  up with  $y[j]$ . In this case, we must have

$$\begin{aligned} \text{opt}[i][j] &= \text{opt}[i+1][j+1] \text{ if } x[i] \text{ equals } y[j], \text{ or} \\ \text{opt}[i][j] &= \text{opt}[i+1][j+1] + 1 \text{ otherwise.} \end{aligned}$$

**Case 2.** The optimal alignment matches  $x[i]$  up with a gap. In this case, we must have

$$\text{opt}[i][j] = \text{opt}[i+1][j] + 2.$$

**Case 3.** The optimal alignment matches  $y[j]$  up with a gap. In this case, we must have

$$\text{opt}[i][j] = \text{opt}[i][j+1] + 2.$$

Using these three cases, we will be able to trace our steps back to the bottom right most cell of the matrix from the top left most cell ([0][0]), where we discovered the final edit distance ([M][N]).

### **What I Learned**

This assignment taught me a few things. I got to use valgrind and learnt how to use it as it determines an algorithm's space usage. Using massif visualizer, I discovered a nice way to visualize valgrind's results as well. See the screenshots section for more information. Also, using the Needleman-Wunsch method to calculate subproblems was quite interesting – something I had not really considered before, so it has given me an insight into other programming methods.

## Screenshot:

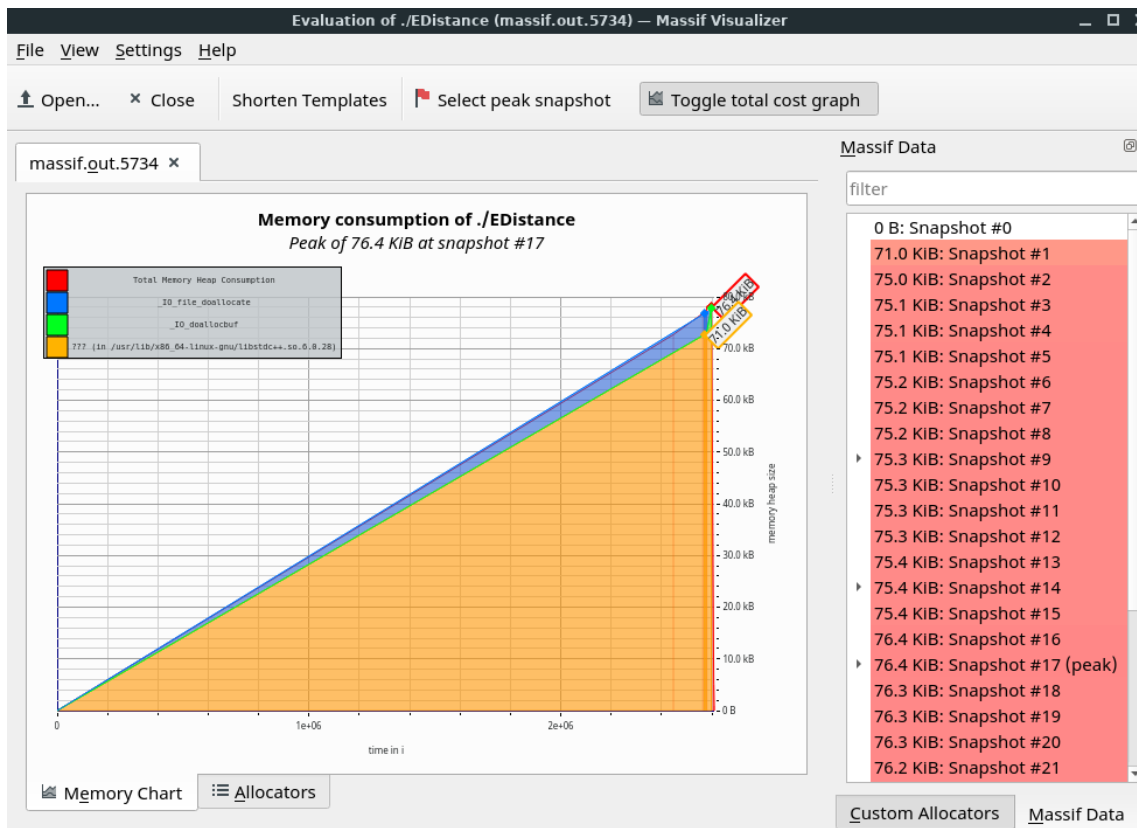
```

Linux Lite Terminal -
File Edit View Terminal Tabs Help
osboxes ~ > PS5 make all
g++ -std=c++11 -c -g -O2 -Wall -Werror -pedantic -o main.o main.cpp
g++ -std=c++11 -c -g -O2 -Wall -Werror -pedantic -o EDistance.o EDistance.cpp
g++ main.o EDistance.o -o EDistance -lsfml-system
osboxes ~ > PS5 ./EDistance <sequence/endgaps7.txt
Edit distance = 4
a - 2
t t 0
a a 0
t t 0
t t 0
a a 0
t t 0
- a 2
Execution time is 0.002366 seconds

osboxes ~ > PS5 valgrind --tool=massif ./EDistance <sequence/endgaps7.txt
==5734== Massif, a heap profiler
==5734== Copyright (C) 2003-2017, and GNU GPL'd, by Nicholas Nethercote
==5734== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==5734== Command: ./EDistance
==5734==
Edit distance = 4
a - 2
t t 0
a a 0
t t 0
t t 0
a a 0
t t 0
- a 2
Execution time is 0.012069 seconds

==5734==

```



**Source code for PS5 DNA Sequence Alignment:****Makefile**

```
1: CXX = g++
2: CXXFLAGS = -std=c++11 -c -g -O2 -Wall -Werror -pedantic
3: OBJ = main.o EDistance.o
4: DEPS = main.cpp EDistance.cpp
5: LIBS = -lsfml-system
6: EXE = EDistance
7:
8: all: $(OBJ)
9:         $(CXX) $(OBJ) -o $(EXE) $(LIBS)
10:
11:         %.o: %.cpp $(DEPS)
12:         $(CXX) $(CXXFLAGS) -o $@ $<
13:
14: clean:
15:         rm $(OBJ) $(EXE)
```

**Main.cpp**

```
1: /*
2: * Copyright 2021 Likhitha Shrinivas Gudalwar
3: *
4: */
5: #include <iostream>
6: #include "EDistance.h"
7: #include <SFML/System.hpp>
8:
9: using namespace std;
10:
11: int main() {
12:     sf::Clock clock;
13:     sf::Time t;
14:     string input_sequence1, input_sequence2;
15:     cin >> input_sequence1 >> input_sequence2;
16:     EDistance ed(input_sequence1, input_sequence2);
17:     auto distance = ed.OptDistance();
18:     auto alignment = ed.Alignment();
19:     t = clock.getElapsedTime();
20:     cout << "Edit distance = " << distance << endl;
21:     cout << alignment << endl;
22:     cout << "Execution time is " << t.asSeconds() << " seconds \n" <<
std::endl;
23:     // cout << "Edit distance = " << distance << endl;
24:     return 0;
25: }
26:
```

**EDistance.h**

```
1: /*
2: * Copyright 2021 Likhitha Shrinivas Gudalwar
3: *
4: */
5: #ifndef EDISTANCE_H
6: #define EDISTANCE_H
7:
8: #include <string>
9: #include<vector>
10: #include <iostream>
11: #include <SFML/System.hpp>
12:
13: class EDistance
14: {
15:
16: private:
17: int **_opt;
18: std::string x;
19: std::string y;
20: int M, N;
21:
22: public:
23: EDistance(const std::string &a, const std::string &b);
24: ~EDistance();
25: static int penalty(char a, char b);
26: static int min(int a, int b, int c);
27: int OptDistance();
28: void PrintMatrix();
29: std::string Alignment();
30: };
31:
32: #endif
```

**EDistance.cpp**

```

1: /*
2: * Copyright 2021 Likhitha Shrinivas Gudalwar
3: *
4: */
5: #include "EDistance.h"
6:
7: EDistance::EDistance(const std::string &a, const std::string &b)
8: {
9: this->x = a;
10: this->y = b;
11: M = x.size();
12: N = y.size();
13: _opt = new int *[M + 1]();
14: if(!_opt)
15: {
16: std::cout << "Not Enough Memeory" << std::endl;
17: exit(1);
18: }
19: for (int i = 0; i < M + 1; i++)
20: _opt[i] = new int[N + 1]();
21: for (int i = 0; i <= M; i++)
22: {
23: _opt[i][N] = [=] { return 2 * (M-i); } ();
24:
25: // _opt[i][N] = 2 * (M - i);
26: }
27: for (int j = 0; j <= N; j++)
28: {
29: _opt[M][j] = [=] { return 2 * (N-j); } ();
30: // _opt[M][j] = 2 * (N - j);
31: }
32: }
33:
34: int EDistance::penalty(char a, char b)
35: {
36: if(a == b) // Equal characters
37: {
38: return 0;
39: }
40:
41: else if(a != b)
42: { // not equal and no spaces
43: return 1;
44: }
45:
46: // If something fails, return a -1.
47: // We can check this for errors.
48:
49: return -1;
50: }
51:
52: int EDistance::min(int a, int b, int c)
53: {
54: if(a < b && a < c)

```



```

55: {
56: return a;
57: }
58:
59: else if(b < a && b < c)
60: {
61: return b;
62: }
63:
64: else if(c < a && c < b)
65: {
66: return c;
67: }
68:
69: // They are all equal if we get here.
70: // So just return a cause w/e
71: return a;
72: }
73:
74: int EDistance::OptDistance()
75: {
76: for (int i = M - 1; i >= 0; i--)
77: for (int j = N - 1; j >= 0; j--)
78: _opt[i][j] = min(_opt[i + 1][j + 1] + penalty(x[i], y[j]), _opt[i + 1][j]
+ 2, _opt[i][j + 1] + 2);
79: return _opt[0][0];
80: }
81:
82: std::string EDistance::Alignment()
83: {
84: std::string result_string;
85: int i = 0, j = 0;
86: while (i < M || j < N) {
87: if (i < M && j < N && x[i] == y[j] && _opt[i][j] == _opt[i + 1][j + 1]) {
88: {
89: result_string += x[i];
90: result_string += " ";
91: result_string += y[j];
92: result_string += " 0";
93: i++;
94: j++;
95: } else
96: if (i < M && j < N && x[i] != y[j] && _opt[i][j] == _opt[i + 1][j + 1] + 1) {
97: result_string += x[i];
98: result_string += " ";
99: result_string += y[j];
100: result_string += " 1";
101: i++;
102: j++;
103: } else if (i < M && _opt[i][j] == _opt[i + 1][j] + 2)
104: {
105: result_string += x[i];
106: result_string += " - 2";
107: i++;
108: } else if (j < N && _opt[i][j] == _opt[i][j + 1] + 2)
109: {
110: result_string += "- ";

```

```
111: result_string += y[j];
112: result_string += " 2";
113: j++;
114: }
115: if(i < M || j < N) result_string += "\n";
116: }
117: return result_string;
118: }
119:
120: EDistance::~~EDistance()
121: {
122: for(int i = 0; i <= M; i++)
123: delete[] _opt[i];
124: delete[] _opt;
125: }
```

## **PS6: Random Writer**

### **About the Assignment**

For PS6, I created a class called “RandWriter” that implements Markov model to analyze an input text for transitions between k-grams (a fixed number of characters) and the following letter. Markov model is a simple mathematical model that counts the number of occurrences and sequences of characters in an English word or sentence. The RandWriter class implements several methods, including freq(), which returns the frequency of a character in a k-gram (fixed number of text) and k\_Rand(), which returns a random character that follows a given k-gram. Using this RandWriter class, we can generate pseudo-random text using statistics – specifically, the generate() method, which returns a string of random characters based on a given k-gram.

### **Key Concepts**

The main idea behind this assignment is to recognize and implement a probabilistic model of the text: given a specific k-gram series of letters, what letters follow at what probabilities? I used a C++ map to create this RandWriter class, which I configured as a std::string, int map – that is, the k-gram is the key to the map, and the int is the value, or number of occurrences of the given k-gram. My RandWriter has a variety of methods including:

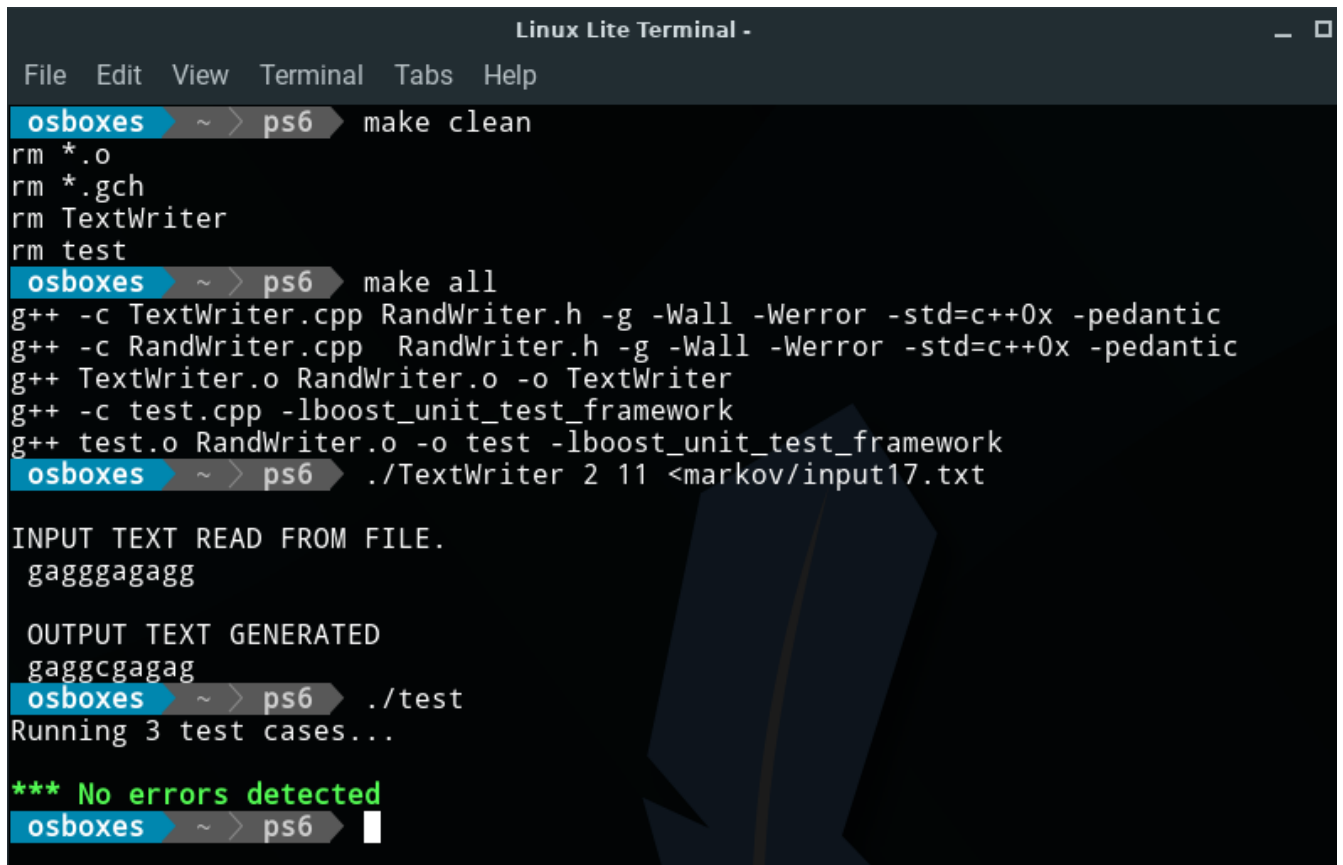
1. freq() – two versions of this method
  - freq(k\_gram) returns the number of times the k-gram was found in the original text.
  - freq(k\_gram,c) returns the number of times the k-gram was followed by the character c in the original text.
2. k\_Rand() – generates a random character that follows the given k-gram
3. generate() – generates a string of length T characters which simulates a trajectory through the given Markov chain.

These four methods were critical to getting the TextWriter program working correctly. The program TextWriter that takes two command-line integers k and L, reads the input text from standard input and builds a Markov model of order k from the input text; then, starting with the

k-gram consisting of the first k characters of the input text, prints out L characters generated by simulating a trajectory through the corresponding Markov chain. Apart from this the test cases in the test.cpp which were used to test the methods in RandWriter were also central to the assignment.

### **What I Learned**

I learned a new concept i.e the Markov chains that are used to create a statistical model of the sequences of letters model. As Markov chains are now widely used in speech recognition, handwriting recognition, information retrieval, data compression, and spam filtering it was pretty fascinating learning about it. Most autocorrect functions on smartphones, for example, use a Markov chain to predict what you're trying to say. It is done by determining the frequencies of characters in the same way that we did, and then using these frequencies to make an educated guess as to what word you are attempting to type / say / etc. It was really cool to figure out how these programs work, and it was also really cool to implement a design that is used in the real world. Inputting text into the finished program is also entertaining because you get completely unexpected results, but these results make sense when you consider how the text is generated – it's random, but it's also based on how the original text was generated. I was also able to get a clear understanding on the lambda expressions by the end of this assignment.

Screenshot:

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
osboxes ~ > ps6 make clean
rm *.o
rm *.gch
rm TextWriter
rm test
osboxes ~ > ps6 make all
g++ -c TextWriter.cpp RandWriter.h -g -Wall -Werror -std=c++0x -pedantic
g++ -c RandWriter.cpp RandWriter.h -g -Wall -Werror -std=c++0x -pedantic
g++ TextWriter.o RandWriter.o -o TextWriter
g++ -c test.cpp -lboost_unit_test_framework
g++ test.o RandWriter.o -o test -lboost_unit_test_framework
osboxes ~ > ps6 ./TextWriter 2 11 <markov/input17.txt

INPUT TEXT READ FROM FILE.
gagggagagg

OUTPUT TEXT GENERATED
gaggcgagag
osboxes ~ > ps6 ./test
Running 3 test cases...

*** No errors detected
osboxes ~ > ps6
```

**Source code for PS6 Random Writer:****Makefile**

```
1: CXX = g++
2: CXXFLAGS = -g -Wall -Werror -std=c++0x -pedantic
3: Boost = -lboost_unit_test_framework
4:
5: all: TextWriter test
6:
7: TextWriter: TextWriter.o RandWriter.o
8:      $(CXX) TextWriter.o RandWriter.o -o TextWriter
9:
10: test: test.o RandWriter.o
11:      $(CXX) test.o RandWriter.o -o test $(Boost)
12:
13: TextWriter.o: TextWriter.cpp RandWriter.h
14:      $(CXX) -c TextWriter.cpp RandWriter.h $(CXXFLAGS)
15:
16: RandWriter.o: RandWriter.cpp RandWriter.h
17:      $(CXX) -c RandWriter.cpp RandWriter.h $(CXXFLAGS)
18:
19: test.o: test.cpp
20:      $(CXX) -c test.cpp $(Boost)
21:
22: clean:
23:      rm *.o
24:      rm *.gch
25:      rm TextWriter
26:      rm test
```

**TextWriter.cpp**

```

1: /*
2:  * Copyright 2021 Likhitha Shrinivas Gudalwar
3:  *
4:  */
5: #include <string>
6: #include "RandWriter.h"
7:
8: int main(int argc, const char* argv[]) {
9:     if (argc != 3) {
10:         std::cout << "Usage: ./TextWriter (int K) (int L)\n";
11:         return 0;
12:     }
13:     std::string str_k(argv[1]);
14:     std::string str_l(argv[2]);
15:
16:     // Get the command line arguments as ints.
17:     /* int k = std::stoi(str_k);
18:     int l = std::stoi(str_l);
19:     */
20:     int _k = std::stoi(str_k);
21:     auto fk = [_k] { return _k; };
22:     auto k = fk();
23:
24:     int _l = std::stoi(str_l);
25:     auto fl = [_l] { return _l; };
26:     auto l = fl();
27:
28:     std::string input = "";
29:     std::string current_txt = "";    // Set these to NULL just to be sure.
30:
31:     // This will read the entire file that you pipe into stdio.
32:     while (std::cin >> current_txt) {
33:         input += " " + current_txt;
34:         current_txt = "";
35:     }
36:
37:     // I figured we should output the user's input for sanity checking.
38:     std::cout << "\n\n INPUT TEXT READ FROM FILE.\n";
39:
40:     // Only show the first L characters that the user cares about though.
41:     for (int a = 0; a < l; a++) {
42:         std::cout << input[a];
43:
44:         // This is for formatting, I just wanted to be able to read the text.
45:         if (input[a] == '.' || input[a] == '!') {
46:             std::cout << "\n";
47:         }
48:     }
49:
50:     // First make a final output string to use.
51:     std::string output_string = "";
52:
53:     // We also need to use a MarkovModel object! Give it the int k as the
54:     // order. Also, the input text will be our string that we pass to the

```

```
55: // constructor.
56: RandWriter amazing(input, k);
57:
58: output_string += "" + amazing.gen(input.substr(0, k), 1);
59:
60: // We're done! Just output the text to standard IO for the user to see.
61: std::cout << "\n\n OUTPUT TEXT GENERATED\n";
62:
63: // Rather than just dump to cout, I figured I'd format this nicely too.
64: // I just added a bunch of newlines.
65: for (int a = 0; a < l; a++) {
66:     std::cout << output_string[a];
67:
68:     // This is for formatting, I just wanted to be able to read the text.
69:     if (output_string[a] == '.' || output_string[a] == '!') {
70:         std::cout << "\n";
71:     }
72: }
73:
74: std::cout << "\n";
75:
76: return 0;
77: }
```



**RandWriter.h**

```

1: /*
2: * Copyright 2021 Likhitha Shrinivas Gudalwar
3: *
4: */
5: #ifndef RANDWRITER_HPP // NOLINT
6: #define RANDWRITER_HPP // NOLINT
7: #include <algorithm>
8: #include <iostream>
9: #include <map>
10: #include <string>
11: #include <stdexcept>
12:
13:
14: class RandWriter {
15: public:
16: /* Creates a Markov model class called
17: *RingWriter of order k from the given text.
18: Assume that the text has a length of at least k. */
19:
20: RandWriter(std::string text, int k);
21:
22: // Order k of Markov model
23: int order_k();
24:
25: // Number of occurrences of kgram in text.
26: // throw an exception if kgram is not of length k
27: int freq(std::string k_gram);
28:
29: /* Number of times that character c follows kgram
30: * If order = 0, return number of times char c appears
31: * ( throw an exception if kgram is not of length k) */
32: int freq(std::string k_gram, char c);
33:
34: /* Random character following given kgram
35: * Throw an exception if kgram is not of length k.
36: * Throw an exception if no such kgram. */
37: char k_Rand(std::string k_gram);
38:
39: /* Generate a string of length T characters by simulating a
40: * trajectory through the corresponding Markov chain.
41: * The first k characters of the newly generated string should be
42: * the argument kgram. ** Assume that L is at least k. **
43: * Throw an exception if kgram is not of length k. */
44: std::string gen(std::string k_gram, int L);
45:
46: /* overload the stream insertion operator and display the internal
47: * state of the Markov Model. Print out the order, the alphabet,
48: * and the frequencies of the k-grams and k+1-grams. */
49: friend std::ostream& operator<< (std::ostream &out, RandWriter &mm);
50:
51: private:
52: int order;
53: std::map <std::string, int> _kgrams;
54: std::string _alphabet;

```

```
55: };  
56: #endif // NOLINT
```

**RandWriter.cpp**

```

1: /*
2:  * Copyright 2021 Likhitha Shrinivas Gudalwar
3:  *
4:  */
5: #include "RandWriter.h"
6: #include <algorithm>
7: #include <map>
8: #include <string>
9: #include <stdexcept>
10: #include <vector>
11: #include <utility>
12:
13: /* Creates a Markov model of order k from the given text.
14:  * Assume that the text has a length of at least k. */
15: RandWriter::RandWriter(std::string text, int k) {
16:     // Set the order.
17:     order = k;
18:
19:     // Seed the random number generator for later.
20:     srand((int)time(NULL)); // NOLINT
21:
22:     // Need to treat the text as circular! So wrap around the first k
23:     // characters. Add the wrap around portion.
24:     std::string text_circular = text; // Make a copy.
25:
26:     for (int a = 0; a < order; a++) {
27:         text_circular.push_back(text[a]);
28:     }
29:
30:     int text_len = text.length(); // Find the text's length, easier later
31:     on.
32:     // Now we need to set the alphabet.
33:     char tmp;
34:     bool inAlphabet = false;
35:
36:     // Go through the entire text and pick out all the individual letters,
37:     for (int i = 0; i < text_len; i++) {
38:         tmp = text.at(i);
39:         inAlphabet = false;
40:
41:         // See if this letter has been added to the alphabet.
42:         for (unsigned int y = 0; y < _alphabet.length(); y++) {
43:             // tmp is already in the alphabet!
44:             // Also ignore upper case,
45:             if (_alphabet.at(y) == tmp) {
46:                 inAlphabet = true; // Match it as being in the alphabet.
47:             }
48:         }
49:
50:         // If tmp isn't in the alphabet, isAlpha should be FALSE.
51:         // So push it back to the alphabet.
52:         if (!inAlphabet) {
53:             _alphabet.push_back(tmp);

```

```

54:     }
55: }
56:
57: // Now that we've got the alphabet, why not sort it alphabetically?
58: std::sort(_alphabet.begin(), _alphabet.end());
59:
60: std::string tmp_str;
61: int x, y;
62:
63: // Do up to text.length() substring comparisons.
64: // This first part just "finds" kgrams and puts a "0" next to them.
65: for (x = order; x <= order + 1; x++) {
66:     // Go through the entire text.
67:     for (y = 0; y < text_len; y++) {
68:         // This collects all given kgrams, and adds a "0" that we can use
69:         // later on to increment. We basically find ALL the kgrams, then
70:         // find their occurrences after.
71:
72:         // current kgram we want.
73:         tmp_str.clear();
74:         tmp_str = text_circular.substr(y, x);
75:
76:         // Insert the 0.
77:         _kgrams.insert(std::pair<std::string, int>(tmp_str, 0));
78:     }
79: }
80:
81: // Need an iterator for going through the kgrams map.
82: std::map<std::string, int>::iterator it;
83: int count_tmp = 0;
84:
85: // Now let's count the kgrams!
86: // Uses same loop as above.
87: for (x = order; x <= order + 1; x++) {
88:     // Go through the entire text.
89:     for (y = 0; y < text_len; y++) {
90:         // Let's get the current kgram we're comparing against.
91:
92:         tmp_str.clear();
93:         tmp_str = text_circular.substr(y, x);
94:
95:         // Now let's get the kgram's current count.
96:         it = _kgrams.find(tmp_str);
97:         count_tmp = it->second;
98:
99:         // Increment the count by 1.
100:         count_tmp++;
101:
102:         // Reinsert the count into the map.
103:         _kgrams[tmp_str] = count_tmp;
104:     }
105: }
106: }
107:
108:
109: // Order k of Markov model
110: int RandWriter::order_k() {

```

```

111:   return order;
112: }
113:
114: /* Number of occurrences of kgram in text.
115:  * -> Throw an exception if kgram is not of length k          */
116: int RandWriter::freq(std::string k_gram) {
117:     // Throw an exception if kgram is not of length k
118:     if (k_gram.length() != (unsigned)order) {
119:         throw
120:             std::runtime_error("Error - kgram not of length k.");
121:     }
122:
123:     // Use std::map::find to see if we can find the kgram.
124:     std::map<std::string, int>::iterator it;
125:     it = _kgrams.find(k_gram);
126:
127:     // If it equals map::end, we didn't find it, so return 0.
128:     if (it == _kgrams.end()) {
129:         return 0;
130:     }
131:
132:     // Other wise return the given kgram since we found it.
133:     return it->second;
134: }
135:
136:
137: /* Number of times that character c follows kgram
138:  * If order = 0, return number of times char c appears
139:  * -> Throw an exception if kgram is not of length k          */
140: int RandWriter::freq(std::string k_gram, char c) {
141:     // Throw an exception if kgram is not of length k
142:     if (k_gram.length() != (unsigned)order) {
143:         throw
144:             std::runtime_error("Error - kgram not of length k.");
145:     }
146:
147:     // use std::map::find to see if we can find the kgram + c.
148:     std::map<std::string, int>::iterator it;
149:     k_gram.push_back(c);
150:     it = _kgrams.find(k_gram);
151:
152:     // If it equals map::end, we didn't find it, so return 0.
153:     if (it == _kgrams.end()) {
154:         return 0;
155:     }
156:
157:     // Other wise return the given kgram since we found it.
158:     return it->second;
159: }
160:
161: /* Returns a random character following the given kgram
162:  * -> Throw an exception if kgram is not of length k.
163:  * -> Throw an exception if no such kgram.                    */
164: char RandWriter::k_Rand(std::string k_gram) {
165:     // Throw an exception if kgram is not of length k.
166:     if (k_gram.length() != (unsigned)order) {
167:         throw std::runtime_error("Error - kgram not of length k (randk)");

```

```

168:     }
169:
170:     // Need an iterator for going through the kgrams map.
171:     std::map<std::string, int>::iterator it;
172:
173:     // Search through and see if we find the given kgram.
174:     it = _kgrams.find(k_gram);
175:
176:     // We didn't find it. Throw an exception.
177:     if (it == _kgrams.end()) {
178:         throw std::runtime_error("Error - Could not find the given kgram!
179: (randk)");
180:     }
181:     // DEBUGGING
182:     //     std::cout << "\n\n\n";
183:
184:     int v = freq(k_gram); // local variable
185:     // copies v into the callable object named f
186:     auto f = [v] { return v; };
187:     auto kgram_freq = f();
188:     // Get the freq of the given kgram. (we will rand by this number!)
189:
190:     //     int kgram_freq = freq(k_gram);
191:
192:     // MORE DEBUG STATEMENTS.
193:     //     std::cout << "Freq of kgram is: " << kgram_freq << "\n";
194:
195:     // Picking a random int from the possible characters.
196:     // This should be an int from 1 to the total number of possible letters.
197:     int random_value = rand() % kgram_freq; // NOLINT
198:
199:     // Debugging
200:     //     std::cout << "Random number value is: " << random_value << "\n";
201:
202:     double test_freq = 0;
203:     double random_num = static_cast<double>(random_value) / kgram_freq;
204:     double last_values = 0;
205:
206:     // Non Bugz
207:     //std::cout <<"Random num as a double/kgram_freq ="<< random_num <<"\n";
208:
209:     // Go through all the letters.
210:     for (unsigned int a = 0; a < _alphabet.length(); a++) {
211:         // This line basically calculates the probability as a double.
212:
213:         test_freq = static_cast<double>(freq(k_gram, _alphabet[a])) / kgram_freq;
214:
215:         // Some debug couts
216:         //std::cout << "Letter: " << _alphabet[a] <<" -> Freq for that letter:";
217:         //     std::cout << test_freq << "\n";
218:
219:         if (random_num < test_freq + last_values && test_freq != 0) {
220:             // Return this letter since it matches.
221:             return _alphabet[a];
222:         }
223:
224:         last_values += test_freq;

```

```

225:     }
226:
227:     // This is here for error checking. We should never reach this point
228:     // unless something in the above code is wrong.
229:     return '-';
230: }
231:
232:
233: /* Generate a string of length L characters by simulating a trajectory
234:  * through the corresponding Markov chain.
235:  * The first k characters of the newly generated string should be
236:  * the argument kgram. ** Assume that L is at least k. **
237:  * -> Throw an exception if kgram is not of length k. */
238: std::string RandWriter::gen(std::string k_gram, int L) {
239:     // Throw an exception if kgram is not of length k.
240:     if (k_gram.length() != (unsigned)order) {
241:         throw std::runtime_error("Error - kgram not of length k. (gen)");
242:     }
243:
244:     // The final string we will return. We'll build it up over time.
245:     std::string final_string = "";
246:
247:     // Temp char for using to collect the return value from randk()
248:     char return_char;
249:
250:     // Add the kgram to it.
251:     final_string += "" + k_gram;
252:
253:     for (unsigned int a = 0; a < (L - (unsigned)order); a++) {
254:         // Call randk on the substring we're looking at.
255:         // Note we want just _order long kgram to compare against.
256:         return_char = k_Rand(final_string.substr(a, order));
257:
258:         // Add the return_char to final_string
259:         final_string.push_back(return_char);
260:
261:         // Keep looping til it stops.
262:     }
263:
264:     // When we get here, we're done. YAY.
265:     return final_string;
266: }
267:
268:
269: /* Overload the stream insertion operator and display the internal
270:  * state of the Markov Model. Print out the order, the alphabet,
271:  * and the frequencies of the k-grams and k+1-grams. */
272: std::ostream& operator<< (std::ostream &out, RandWriter &mm) {
273:     out << "\n_Order: " << mm.order << "\n";
274:     out << "Alphabet: " << mm._alphabet << "\n";
275:
276:     out << "Kgrams map: \n\n";
277:
278:     std::map<std::string, int>::iterator it;
279:
280:     for (it = mm._kgrams.begin(); it != mm._kgrams.end(); it++) {
281:         out << it->first << "\t" << it->second << "\n";

```

```
282:  }  
283:  
284:  return out;  
285: }
```



**Test.cpp**

```

1: /*
2:  * Copyright 2021 Likhitha Shrinivas Gudalwar
3:  *
4:  */
5: #include <iostream>
6: #include <string>
7: #include <exception>
8: #include <stdexcept>
9:
10: #include "RandWriter.h"
11:
12: #define BOOST_TEST_DYN_LINK
13: #define BOOST_TEST_MODULE Main
14: #include <boost/test/unit_test.hpp>
15:
16: using namespace std; // NOLINT
17:
18: BOOST_AUTO_TEST_CASE(order0) {
19:     // normal constructor
20:     BOOST_REQUIRE_NO_THROW(RandWriter("gagggagaggcgagaaa", 0));
21:
22:     RandWriter mm("gagggagaggcgagaaa", 0);
23:
24:     BOOST_REQUIRE(mm.order_k() == 0);
25:     BOOST_REQUIRE(mm.freq("") == 17); // length of input in constructor
26:     BOOST_REQUIRE_THROW(mm.freq("x"), std::runtime_error);
27:
28:     BOOST_REQUIRE(mm.freq("", 'g') == 9);
29:     BOOST_REQUIRE(mm.freq("", 'a') == 7);
30:     BOOST_REQUIRE(mm.freq("", 'c') == 1);
31:     BOOST_REQUIRE(mm.freq("", 'x') == 0);
32: }
33:
34: BOOST_AUTO_TEST_CASE(order1) {
35:     // normal constructor
36:     BOOST_REQUIRE_NO_THROW(RandWriter("gagggagaggcgagaaa", 1));
37:
38:     RandWriter mm("gagggagaggcgagaaa", 1);
39:
40:     BOOST_REQUIRE(mm.order_k() == 1);
41:     BOOST_REQUIRE_THROW(mm.freq(""), std::runtime_error);
42:     BOOST_REQUIRE_THROW(mm.freq("xx"), std::runtime_error);
43:
44:     BOOST_REQUIRE(mm.freq("a") == 7);
45:     BOOST_REQUIRE(mm.freq("g") == 9);
46:     BOOST_REQUIRE(mm.freq("c") == 1);
47:
48:     BOOST_REQUIRE(mm.freq("a", 'a') == 2);
49:     BOOST_REQUIRE(mm.freq("a", 'c') == 0);
50:     BOOST_REQUIRE(mm.freq("a", 'g') == 5);
51:
52:     BOOST_REQUIRE(mm.freq("c", 'a') == 0);
53:     BOOST_REQUIRE(mm.freq("c", 'c') == 0);
54:     BOOST_REQUIRE(mm.freq("c", 'g') == 1);

```

```

55:
56: BOOST_REQUIRE(mm.freq("g", 'a') == 5);
57: BOOST_REQUIRE(mm.freq("g", 'c') == 1);
58: BOOST_REQUIRE(mm.freq("g", 'g') == 3);
59:
60: BOOST_REQUIRE_NO_THROW(mm.k_Rand("a"));
61: BOOST_REQUIRE_NO_THROW(mm.k_Rand("c"));
62: BOOST_REQUIRE_NO_THROW(mm.k_Rand("g"));
63:
64: BOOST_REQUIRE_THROW(mm.k_Rand("x"), std::runtime_error);
65:
66: BOOST_REQUIRE_THROW(mm.k_Rand("xx"), std::runtime_error);
67: }
68:
69: BOOST_AUTO_TEST_CASE(order2) {
70:     // normal constructor
71:     BOOST_REQUIRE_NO_THROW(RandWriter("gagggagagggcgagaaa", 2));
72:
73:     RandWriter mm("gagggagagggcgagaaa", 2);
74:
75:     BOOST_REQUIRE(mm.order_k() == 2);
76:
77:     BOOST_REQUIRE_THROW(mm.freq(""), std::runtime_error);
78:     BOOST_REQUIRE_THROW(mm.freq("x"), std::runtime_error);
79:     BOOST_REQUIRE_NO_THROW(mm.freq("xx"));
80:     BOOST_REQUIRE_THROW(mm.freq("", 'g'), std::runtime_error); // wrong
                                                                    length
81:     BOOST_REQUIRE_THROW(mm.freq("x", 'g'), std::runtime_error); // wrong
                                                                    length
82:     BOOST_REQUIRE_THROW(mm.freq("xxx", 'g'), std::runtime_error); // wrong
                                                                    length
83:
84:
85:     BOOST_REQUIRE(mm.freq("aa") == 2);
86:     BOOST_REQUIRE(mm.freq("aa", 'a') == 1);
87:     BOOST_REQUIRE(mm.freq("aa", 'c') == 0);
88:     BOOST_REQUIRE(mm.freq("aa", 'g') == 1);
89:
90:     BOOST_REQUIRE(mm.freq("ag") == 5);
91:     BOOST_REQUIRE(mm.freq("ag", 'a') == 3);
92:     BOOST_REQUIRE(mm.freq("ag", 'c') == 0);
93:     BOOST_REQUIRE(mm.freq("ag", 'g') == 2);
94:
95:     BOOST_REQUIRE(mm.freq("cg") == 1);
96:     BOOST_REQUIRE(mm.freq("cg", 'a') == 1);
97:     BOOST_REQUIRE(mm.freq("cg", 'c') == 0);
98:     BOOST_REQUIRE(mm.freq("cg", 'g') == 0);
99:
100:    BOOST_REQUIRE(mm.freq("ga") == 5);
101:    BOOST_REQUIRE(mm.freq("ga", 'a') == 1);
102:    BOOST_REQUIRE(mm.freq("ga", 'c') == 0);
103:    BOOST_REQUIRE(mm.freq("ga", 'g') == 4);
104:
105:    BOOST_REQUIRE(mm.freq("gc") == 1);
106:    BOOST_REQUIRE(mm.freq("gc", 'a') == 0);
107:    BOOST_REQUIRE(mm.freq("gc", 'c') == 0);
108:    BOOST_REQUIRE(mm.freq("gc", 'g') == 1);

```

```
109:
110: BOOST_REQUIRE(mm.freq("gg") == 3);
111: BOOST_REQUIRE(mm.freq("gg", 'a') == 1);
112: BOOST_REQUIRE(mm.freq("gg", 'c') == 1);
113: BOOST_REQUIRE(mm.freq("gg", 'g') == 1);
114: }
```

## **PS7: Kronos Time Clock**

### **About the Assignment**

This assignment required us to parse a log file from a Kronos InTouch time clock using regular expressions. We used Boost's regular expression library (regex) and date/time libraries to parse the file. This enabled us to create an output file that verifies whether a time clock booted successfully or failed to boot.

### **Key Concepts**

The main concept of this assignment was regular expressions, specifically how to use Boost's regex library to find matches in a string. Apart from this the date/time libraries were central concepts of this assignment which also perform calculations on them. Boost's regex library was simple to use – simply using `boost::regex match` or `boost::regex search` yielded a match against the regular expressions I created. I used the following expressions to match start boots and successful completions:

```
r1 = regex(R"((.*): (\(log.c.166\) server started.*))");
```

This is my regex to detect the boot statement. It finds a matching line with a time stamp, following by the words "(log.c.166) server started". By matching the regex against the entire statement, it was simple to extract the date and time from the current line.

```
r2 = regex("(.*)\.\.\d*:INFO:oejs.AbstractConnector:Started "
           "SelectChannelConnector@0.0.0.0:9080.*");
```

I used this regex to find a succesful boot. A successful boot is marked by a time stamp, followed by the string: "INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080". As with the boot statement, I pulled the date / time from the smatch variable that I passed to the regex search / match methods.

## **What I Learned**

I learned about regular expressions in my Foundations of Computer Science class but never used them in practice. Working on this program provided me with the opportunity to put them to use, and I learned how to use them to parse a program. It was also useful to learn about date and time. After using Boost's date and time libraries, I'm confident in using other date and time libraries in the future. It has provided me with a solid foundation for working with dates that can be helpful to me in future.

**Screenshot:**

```

Linux Lite Terminal -
File Edit View Terminal Tabs Help
osboxes ~ > ps7 make clean
rm *.o
rm main
osboxes ~ > ps7 make all
g++ -c main.cpp -g -Wall -Werror -std=c++0x -pedantic
g++ main.o -o main -lboost_regex -lboost_date_time
osboxes ~ > ps7 ./main device1_intouch.log
osboxes ~ > ps7

```

Open	device1_intouch.log.rpt	Save				
	~/ps7					
1	435369 (log.c.166)	server started	2014-03-25	19:11:59	success	183000ms
2	436500 (log.c.166)	server started	2014-03-25	19:29:59	success	165000ms
3	440719 (log.c.166)	server started	2014-03-25	22:01:46	success	161000ms
4	440866 (log.c.166)	server started	2014-03-26	12:47:42	success	167000ms
5	442094 (log.c.166)	server started	2014-03-26	20:41:34	success	159000ms
6	443073 (log.c.166)	server started	2014-03-27	14:09:01	success	161000ms

*Note: The above file is generated as output when the command that is run on the terminal*

**Source code for PS7 Kronos Time Clock:****Makefile**

```
1: CXX = g++
2: CXXFLAGS = -g -Wall -Werror -std=c++0x -pedantic
3: Boost = -lboost_regex -lboost_date_time
4:
5: all: main
6:
7: main: main.o
8:         $(CXX) main.o -o main $(Boost)
9:
10: main.o: main.cpp
11:         $(CXX) -c main.cpp $(CXXFLAGS)
12:
13: clean:
14:         rm *.o
15:         rm main
```

**Main.cpp**

```

1: /*
2: * Copyright 2021 Likhitha Shrinivas Gudalwar
3: *
4: */
5: #include <iostream>
6: #include <string>
7: #include <fstream>
8: #include <boost/regex.hpp>
9: #include "boost/date_time/posix_time/posix_time.hpp"
10:
11:
12: using std::cout;
13: using std::cin;
14: using std::endl;
15: using std::string;
16:
17: using boost::regex;
18: using boost::smatch;
19: using boost::regex_error;
20:
21: using boost::gregorian::date;
22: using boost::gregorian::from_simple_string;
23: using boost::gregorian::date_period;
24: using boost::gregorian::date_duration;
25:
26: using boost::posix_time::ptime;
27: using boost::posix_time::time_duration;
28:
29:
30: int main(int argc, char **args) {
31: if (argc != 2) {
32: // to make sure the user knows how to give input for the log parser
33: cout << "usage: ./ps7 [logfile]" << endl;
34: exit(1);
35: }
36: string s, rs;
37: regex r1;
38: regex r2;
39: bool flag = false;
40: ptime t1, t2;
41: // Some strings we need. File names, other stuff.
42: string filename(args[1]);
43: std::ifstream infile(filename);
44: std::ofstream outfile(filename + ".rpt");
45: if (!infile || !outfile) {
46: cout << "open file error" << endl;
47: exit(1);
48: }
49: try {
50: // Need to match against something like this:
51: // Start of boot: 2014-02-01 14:02:32: (log.c.166) server started
52: // Success if we find:
53: // "2014-01-26 09:58:04.362:INFO:oejs.AbstractConnector:Started
54: // SelectChannelConnector@0.0.0.0:9080"

```



```

55:
56: // Make two regexes
57:
58: r1 = regex(R"((.*): (\(log.c.166\) server started.*))");
59: r2 = regex("(.*)\\.\\.\\d*:INFO:oejs.AbstractConnector:Started "
60: "SelectChannelConnector@0.0.0.0:9080.*");
61: } catch (regex_error &exc) {
62: cout << "Regex constructor failed with code " << exc.code() << endl;
63: exit(1);
64: }
65: // START AT LINE 1
66: int line_number = 1;
67: string str;
68: while (getline(infile, s)) {
69: if (regex_match(s, r1)) {
70: smatch sm;
71: regex_match(s, sm, r1);
72: if (flag) {
73: outfile << "failure" << endl;
74: }
75: flag = true;
76: t1 = ptime(boost::posix_time::time_from_string(sm[1]));
77: str = sm[2];
78: outfile << line_number << " (log.c.166) server started "
79: << sm[1] << " ";
80: }
81: if (regex_match(s, r2)) {
82: smatch sm;
83: regex_match(s, sm, r2);
84: t2 = ptime(boost::posix_time::time_from_string(sm[1]));
85: time_duration td = t2 - t1;
86: outfile << "success " << td.total_milliseconds()
87: << "ms" << endl;
88: flag = false;
89: }
90: line_number++;
91: }
92: if (flag) {
93: outfile << "failure" << endl;
94: }
95: infile.close();
96: outfile.close();
97: return 0;
98: }

```