Raul Olivares
COMP IV: Project Portfolio
Fall 2021

**Contents:**



Total time to complete: 122 hrs.

# PS0: Hello World

## The Assignment

Hello World was my first Computing IV assignment. The main goal of this assignment was to setup a Linux build environment and to test out the SFML audio/graphics library. This included getting Linux running through a Virtualbox and running some SFML example code to gain experience by testing out different SFML graphic codes. After being given a demo code to work with, we were asked to add some additional features to it. I was able to create a working program that created an image that responded to different keystrokes, switched images, and was able to move around the screen.

## Key Concepts

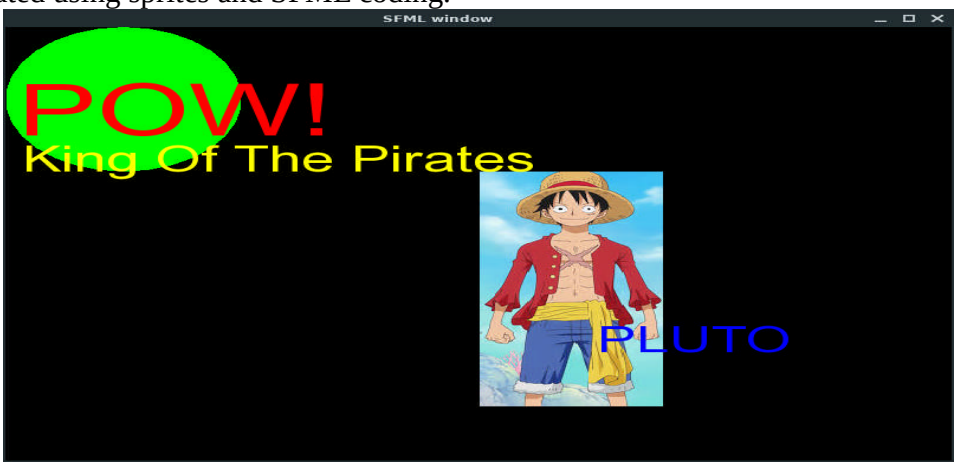
The main concept of this assignment was to get accustomed to SFML and its different classes, methods and uses; which included Images, Sprites, Text, Texture, and Keyboard. Had to get accustomed to how the classes such as Texture, Image, interacted with each other. This is seen in how it is necessary to load an image into a Texture and how a sprite must set a Texture to be used. It was important to learn the ways that the keyboard could affect the sprite so that it would be able to move around the screen through presses of a key.

**Code from main.cpp for sprite**

```
  sf::Texture texture;  //create texture object
 if (!texture.loadFromFile("sprite.png"))  // Load a sprite to display
   return EXIT_FAILURE;
 sf::Sprite sprite(texture);   //load texture image into sprite
sprite.setPosition(sf::Vector2f(300,200));//set starting position for sprite
```

## What I Learned

I learned what VirtualBox was, how to use it, and what virtualization software was exactly. I had previously only used a pre-existing VM, but now learned how to create and run one. I was able to expand my knowledge on Linux, as well as learning how to settup build environments for Linux. I learned what SFML was and how to use it in this assignment. Some of the things I learned in SFML were: how to display images in an SFML window, how to set up and control sprites using SFML's Keyboard library, displaying text with SFML, plus some other basic SFML knowledge. This assignment also sparked my interest in learning how animations are created using sprites and SFML coding.



```
//*******Main.cpp Main class file
// Created by: Raul Olivares
// On: September 7, 2021
// Assignment: PS0
// Teacher Dr. Rykalova
// Class: COMP 2040 HY 1 201
//Program is meant to run on SFML and display a window with a green circle, a text, and a moving sprite that responds
//to different keystrokes from the keyboard
// Bugs: 1.The sprite image will slowly disappear off screen if you dont use the keyboard keys to stop it and prevent its exit.
//   2. One of the loaded images does not appear for some reason?
//    3. code commented out that would make sprite constantly move in a circle because code runs too quick in iteration
//      and you cant visually see it happening.

#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>  //including files needed to successfully run program

int main()     //main function
{
    // Create the main window
    sf::RenderWindow window(sf::VideoMode(800, 600), "SFML window");

    window.setFramerateLimit(10);   //set speed of frame
    sf::CircleShape shape(100.f);    //create a circle object
    shape.setFillColor(sf::Color::Green);//set color green for circle object

    sf::Texture texture;  //create texture object
     if (!texture.loadFromFile("sprite.png"))  // Load a sprite to display
        return EXIT_FAILURE;
    sf::Sprite sprite(texture);   //load texture image into sprite
    sprite.setPosition(sf::Vector2f(300,200));//set starting position for sprite

     sf::Texture texture1;
    if (!texture1.loadFromFile("luffy1.jpeg"))            //load second sprite image
        return EXIT_FAILURE;
    sf::Sprite sprite1(texture1);
       sprite1.setPosition(sf::Vector2f(300,200));

     sf::Texture texture2;
    if (!texture2.loadFromFile("luffy2.jpeg"))
    //if (!texture2.loadFromFile("sprite.png"))
        return EXIT_FAILURE;
    sf::Sprite sprite2(texture);                          //load third sprite image
    sprite2.setPosition(sf::Vector2f(300,200));

    sf::Texture texture3;
    if (!texture3.loadFromFile("luffy3.jpeg"))
        return EXIT_FAILURE;
    sf::Sprite sprite3(texture3);                    //load fourth sprite image
    sprite3.setPosition(sf::Vector2f(300,200));

    sf::Texture texture4;
    if (!texture4.loadFromFile("sprite.png"))
        return EXIT_FAILURE;
```

```
    sf::Sprite sprite4(texture4); //load fifth sprite image that will be use
d to hold other sprite image so that we can change between images
     sprite4.setPosition(sf::Vector2f(300,200));

    // Create a graphical text to display
    sf::Font font;

    if (!font.loadFromFile("arial.ttf"))
        return EXIT_FAILURE;
          sf::Text text("POW!", font, 100);              //create text usin
g font for "pow" and determine its different attributes
          text.setFillColor(sf::Color::Red);
          text.setPosition(10.f, 50.f);

           sf::Font font1;
        if (!font1.loadFromFile("arial.ttf"))
        return EXIT_FAILURE;
          sf::Text text1("King Of The Pirates", font, 50);     //create text
 using font for "pow" and determine its different attributes
          text1.setFillColor(sf::Color::Yellow);
          text1.setPosition(14.f, 150.f);

                 sf::Font font2; //create font object
        if (!font2.loadFromFile("arial.ttf")) //load font information from a
rial.ttf
        return EXIT_FAILURE;
          sf::Text text2("PLUTO", font, 50);//create text saying pluto and d
etermine the size i want of the font
          text2.setFillColor(sf::Color::Blue);//set text2 to blue
          text2.setPosition(500.f, 400.f); //set the position placement of t
ext

    // Start the game loop
   while (window.isOpen())
    {
        sf::Event event; //create event object to allow events to close wind
ow
        while (window.pollEvent(event))
        {
            // Close window: exit
            if (event.type == sf::Event::Closed)
                window.close();
        }
         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))//if statement to
see if key is pressed to create event to move sprite
               { sprite.move(sf::Vector2f(0, -5));  //changes all positions
of the sprite up by one when up keyboard is pressed
                sprite1.move(sf::Vector2f(0, -5));
                 sprite2.move(sf::Vector2f(0, -5));
                  sprite3.move(sf::Vector2f(0, -5));
                   sprite4.move(sf::Vector2f(0, -5));
                //sprite.setPosition(sprite1.getPosition));
                // sprite4=sprite2;
                //sprite3=sprite4;
            sprite4=sprite;

            sprite=sprite1;

            sprite1=sprite2;
                sprite2=sprite3;
                sprite3=sprite4;
```

```
  106:
  107:
  108:
  109:
  110:
  111:                }
  112:
  113:                 if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
  114:                {sprite.move(sf::Vector2f(0, 5));
  115:                sprite1.move(sf::Vector2f(0, 5));   //changes all positions
of the sprite down by one when down keyboard is pressed
  116:                sprite2.move(sf::Vector2f(0, 5));
  117:                  sprite3.move(sf::Vector2f(0, 5));
  118:                   sprite4.move(sf::Vector2f(0, 5));
  119:               // sprite2=sprite;
  120:               //sprite=sprite1;
  121:               // sprite1=sprite2;
  122:               sprite4=sprite;
  123:
  124:            sprite=sprite1;
  125:
  126:            sprite1=sprite2;
  127:                sprite2=sprite3;
  128:                sprite3=sprite4;
  129:
  130:                }
  131:                 if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
  132:                {sprite.move(sf::Vector2f(5, 0));
  133:                sprite1.move(sf::Vector2f(5, 0));   //changes all positions
of the sprite to the right by one when right keyboard is pressed
  134:                sprite2.move(sf::Vector2f(5, 0));
  135:                  sprite3.move(sf::Vector2f(5, 0));
  136:                   sprite4.move(sf::Vector2f(5, 0));
  137:                // sprite4=sprite;
  138:               // sprite=sprite3;
  139:                //sprite3=sprite4;
  140:                }
  141:                 if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
  142:               { sprite.move(sf::Vector2f(-5, 0));
  143:                sprite1.move(sf::Vector2f(-5, 0));
  144:                sprite2.move(sf::Vector2f(-5,0));   //changes all positions
of the sprite to the left by one when left keyboard is pressed
  145:                  sprite3.move(sf::Vector2f(-5,0));
  146:                   sprite4.move(sf::Vector2f(-5,0));
  147:                }
  148:
  149:         // Clear screen
  150:            window.clear();
  151:         window.draw(sprite);
  152:      window.draw(shape);//draw the green circle
  153:         //window.draw(sprite);//draw the sprite and other sprite images that
 rotate as sprite
  154:            window.draw(text);
  155:           // bool movement = true;
  156:      //bool up = false;
  157:
  158:     //bool down = false;
  159:      //bool left = false;
  160:
  161:       //while(movement==true){
  162:       //if(movement==true){
```

```
        //  for(float i = 0; i < 4; i++){

          //   sprite.move(sf::Vector2f(2, 0));

            // sprite1.move(sf::Vector2f(2, 0));

         //    sprite2.move(sf::Vector2f(2, 0));

       //      sprite3.move(sf::Vector2f(2, 0));

        //     sprite4.move(sf::Vector2f(2, 0));

        //     if(i==3){movement=false;
        //     up=true;
        //     }
         //    }}
         //    else {}
       //      if(movement==false && up==true){
       //      for(float j = 0; j < 4; j++){

         //    sprite.move(sf::Vector2f(0, 2));

       //      sprite1.move(sf::Vector2f(0, 2));

         //    sprite2.move(sf::Vector2f(0, 2));

        //     sprite3.move(sf::Vector2f(0, 2));

        //     sprite4.move(sf::Vector2f(0, 2));

      //       if(j==3){up=false;
     //        left=true;}}}

       //       else {}
      //        if(movement==false && left==true){
     //        for(float k = 0; k < 4; k++){

       //      sprite.move(sf::Vector2f(-2, 0));

         //    sprite1.move(sf::Vector2f(-2, 0));

     //         sprite2.move(sf::Vector2f(-2, 0));

     //         sprite3.move(sf::Vector2f(-2, 0));

     //         sprite4.move(sf::Vector2f(-2, 0));

       //      if(k==3){left=false;
         //    down=true;}}}

           //   else {}
           //   if(movement==false && down==true){
        //     for(float l = 0; l < 4; l++){

         //    sprite.move(sf::Vector2f(0, -2));

       //      sprite1.move(sf::Vector2f(0, -2));

      //       sprite2.move(sf::Vector2f(0, -2));

      //       sprite3.move(sf::Vector2f(0, -2));
```

```
  224:
  225:       //       sprite4.move(sf::Vector2f(0, -2));
  226:
  227:      //        if(l==3){down=false;
  228:      //        movement=true;}}}
  229:
  230:
  231:       //          break ;}
  232:
  233:                 for(float r = 0; r < 4; r++){ //creates a loop to make image
 move to the right 3 times but since game loops keeps looping
  234:                                          //it will keep looping on itself an
d always have the sprite moving to the left
  235:              sprite.move(sf::Vector2f(.5, 0));
  236:
  237:              sprite1.move(sf::Vector2f(.5, 0));
  238:
  239:              sprite2.move(sf::Vector2f(.5, 0)); //moves sprites by .5 so tha
t we can still move with keyboard to the left
  240:                                                 // because keyboard method m
oves by 1 so it overpowers this to be able to move sprite
  241:                                                 //to the left
  242:              sprite3.move(sf::Vector2f(.5, 0));
  243:
  244:              sprite4.move(sf::Vector2f(.5, 0));}
  245:
  246:         // Draw the strings
  247:         window.draw(text1);
  248:            window.draw(text2);
  249:         // Update the window
  250:         window.display();
  251:     }
  252:     return EXIT_SUCCESS;  //exits program
  253: }
  254:
```

# PS1a: Linear Feedback Shift Register

## The Assignment

This assignment required us to implement a Fibonacci Linear Feedback Shift Register. This type of register shifts all bits left one position and replaces the vacated bit by the exclusive or of the bit shifted off and the bit previously at a given tap position in the register. I was shown that generally, a LFSR has three parameters that characterize the sequence of bits it produces: the number of bits N, the initial seed (the sequence of bits that initializes the register), and the tap position tap. Our main goals were to implement the shift register in a class called “LFSR” using 3 different tap positions and to implement several unit tests using the Boost test framework.

## Key Concepts

I used the Boost test framework to test my LFSR class. The Boost test framework was used to test our LFSR class, by using Boost’s auto rest case method’s to test the step and generate methods against different cases. Another important concept was the use of Makefile to compile, link,create object code, and executable code for the files needed for the program without having to manually enter everything each time. Another important concept was the formula used to shift the register my xoring the designated bits and placing the final bit at the right of the register.

**Code for boost case**

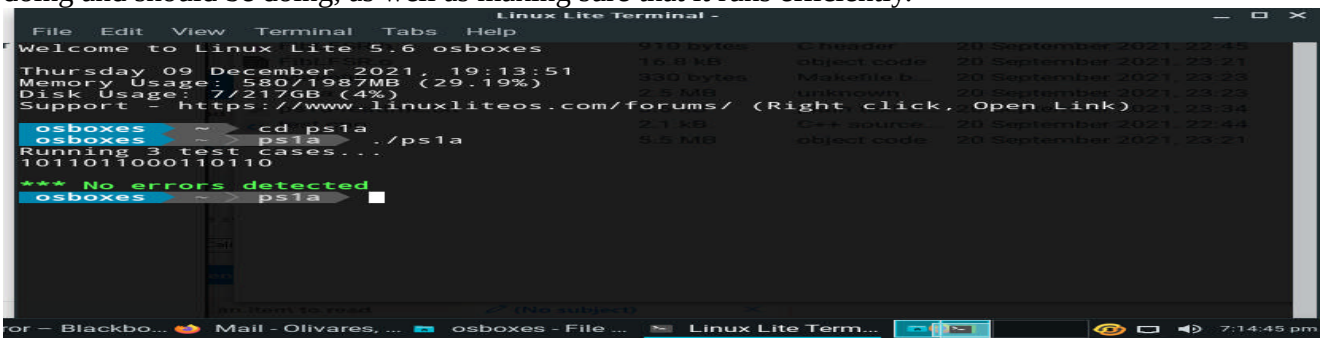
```
BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) { //given test case to make sure function runs appropriately

FibLFSR l("1011011000110110");
BOOST_REQUIRE(l.step() == 0);
```

**Code for makefile**

```
test.o: test.cpp
g++ -std=c++11 -g -Og -Wall -Werror -pedantic -c test.cpp
```

## What I Learned

This assignment taught me about testing in C++ and what Boost tests are and how to implement them. I had never heard about Boost or thought about how important it might be to test individual functions of my code to check that they were running correctly. My usual approach was just to debug and make sure that my program compiled and executed what it was made to do. The idea of unit testing the important functions, or all functions makes a lot of sense to me now; This Boost testing will help me to prevent future bugs from occurring by simulating different possible outcomes in my code. It’s a very effective means of knowing what your code is doing and should be doing, as well as making sure that it runs efficiently.



```
all: ps1a

ps1a: test.o FibLFSR.o
        g++ -std=c++11 -g -Og -Wall -Werror -pedantic -o ps1a test.o FibLFSR
.o -l boost_unit_test_framework


FibLFSR.o : FibLFSR.cpp FibLFSR.h
        g++ -std=c++11 -g -Og -Wall -Werror -pedantic -c FibLFSR.cpp

test.o: test.cpp
        g++ -std=c++11 -g -Og -Wall -Werror -pedantic -c test.cpp
clean:
        rm *.o ps1a
```

```
 1: //Asignment: ps1a
 2: //test.cpp file
 3: //File containg code to make sure LFSR register works
 4: //Created on 9/15/2021
 5: //By Raul Olivares
 6: //Due on :9/20/2021
 7: //Class: COMP 2040 HY 1 201
 8: //Professor:Dr. Rykalova
 9: #include <iostream>
10: #include <string>
11: #include "FibLFSR.h"
12:
13: #define BOOST_TEST_DYN_LINK
14: #define BOOST_TEST_MODULE Main
15: #include <boost/test/included/unit_test.hpp>
16:
17: BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) { //given test case to make sure
function runs appropriately
18:
19:   FibLFSR l("1011011000110110");
20:   BOOST_REQUIRE(l.step() == 0);
21:   BOOST_REQUIRE(l.step() == 0);
22:   BOOST_REQUIRE(l.step() == 0);
23:   BOOST_REQUIRE(l.step() == 1);
24:   BOOST_REQUIRE(l.step() == 1);
25:   BOOST_REQUIRE(l.step() == 0);
26:   BOOST_REQUIRE(l.step() == 0);
27:   BOOST_REQUIRE(l.step() == 1);
28:
29:   FibLFSR l2("1011011000110110");
30:   BOOST_REQUIRE(l2.generate(5) == 3);
31:   FibLFSR lfsr{"1011011000110110"};
32: std::cout<<lfsr<<std::endl;
33:
34:
35:
36:
37: }
38:
39: BOOST_AUTO_TEST_CASE(pluto) {    //test case to check if code is running app
ropriately given different instances
40:
41:
42:   FibLFSR l2("1011011000110110");
43:
44:   FibLFSR lfsr{"1011011000110110"};
45:
46: for(int x{};x<16;x++){
47: BOOST_REQUIRE(lfsr.step() ==l2.step()); //checks if l2 and lfsr have same va
lues or not during step function
48: }
49: BOOST_REQUIRE(lfsr.generate(9)==l2.generate(9)); //checks if different insta
nces using generate function give similar values
50: }
51:
52:
53: BOOST_AUTO_TEST_CASE(thirdBoost) {     //test case if given different seed v
alue, does the function still work how its supposed to
54:
55:   FibLFSR l("1111000011110000");
56:   BOOST_REQUIRE(l.step() == 1);   //checks value in step function against th
```

```
e value 1 that its supossed to have
   57:   BOOST_REQUIRE(l.step() == 0);
   58:   BOOST_REQUIRE(l.step() == 1);
   59:   BOOST_REQUIRE(l.step() == 0);
   60:   BOOST_REQUIRE(l.step() == 1);
   61:   BOOST_REQUIRE(l.step() == 0);
   62:   BOOST_REQUIRE(l.step() == 1);
   63:   BOOST_REQUIRE(l.step() == 0);
   64:
   65:   FibLFSR l2("1111000011110000");
   66:   BOOST_REQUIRE(l2.generate(9) == 341);//checks if generate function value r
eturn matches the value that the function is supposed to return
   67:
   68: }
   69:
```

```
 1: //Asignment: ps1a
 2: //FibLFSR.cpp file
 3: //File containg header code for LFSR register
 4: //Created on 9/15/2021
 5: //By Raul Olivares
 6: //Due on :9/20/2021
 7: //Class: COMP 2040 HY 1 201
 8: //Professor:Dr. Rykalova
 9: #ifndef FIB_LFSR_H
10: #define FIB_LFSR_H
11:
12: #include<string>
13: #include<bitset>
14: #include<iostream>
15:
16: class FibLFSR
17: {
18:
19: public:
20:
21: FibLFSR(std::string seed); //constructor
22: ~FibLFSR();
23:
24: int step(); //step method to produce LFSR
25:
26: int generate(int k);    //generate method to produce variable containing val
ue after doubling and adding new extracted number
27:
28: friend std::ostream& operator<< (std::ostream& os, const FibLFSR lfsr); //fr
iend of ostream operations
29:
30: private:
31:
32: static const size_t ArraySize{16}; // variable used to make a set size for t
he array we need to create the register
33:
34: std::bitset<ArraySize> Register{};  //we place ArraySize in here because bit
set requires a fixed size in advance to initiate
35:
36: };
37: #endif
```

```
//Asignment: ps1a
//FibLFSR.cpp file
//File containg detailed code needed to run the LFSR register which shifts every bit
//to the left, then takes the xor value of the left most bit against 3 taps located in different part of the
//array and then returns the value, as well as placing it into the rightmost bit after the shift. Also
//performing a generate function that sets a variable initiated at 0, then performs shift operation, doubles the variable and adds
//the extracted xor value to it and then returns it.
//Created on 9/15/2021
//By Raul Olivares
//Due on :9/20/2021
//Class: COMP 2040 HY 1 201
//Professor:Dr. Rykalova
#include "FibLFSR.h"
#include<string>       //all included files needed
#include<iostream>
using namespace std; //used if we dont want to write std:: before certain codes


FibLFSR::FibLFSR(std::string seed){

    if(seed.length()>16 || seed.length()<15)  //checks the size of the seed value
   {cout<<"Seed must be 16 bits long."<<endl;}
        else{}

        for(size_t i{} ;i<seed.length();i++)
   {char bit{seed[i]};

        if(bit !='1' && bit !='0')
{cout<<"Register must use 0 & 1 binary code!!!" <<endl;}
// if to make sure only 1's and 0's were used.

Register.set(15 - i, bit =='1');  //sets the register value to hold seed value given
}
}



int FibLFSR ::step()
{
    int output = Register[15] ^ Register[13] ^ Register[12] ^ Register[10] ; //output after xoring and the taps
    Register <<= 1;   //shift operation to move all bits in register by 1 to the left
    Register.set(0, static_cast<bool>(output));   //sets the output of the xor operations into the oth place in reg
    return output;   //returns output

}

int FibLFSR:: generate(int k)
{
    uint16_t generate_output{};//initializing unsigned variable to hold value for generate input
```

```
   51:    for(int i{};i<k;i++) {//for loop
   52:     generate_output <<=1;    //shifting everything to the left
   53:     generate_output |= step();} //sets all the bits into generateoutput afte
r running step function
   54:
   55: return generate_output;  //returns generate_output
   56: }
   57:
   58: std::ostream& operator<<(std::ostream& os, const FibLFSR lfsr)
   59: {
   60: std::string bits;
   61: bits.reserve(FibLFSR::ArraySize);  //reserves space to store the bits
   62: for(int i{FibLFSR::ArraySize -1}; i>=0;i--)
   63: {
   64: bits.push_back('0'+lfsr.Register[i]);//pushes onto bits the values in the re
gister to create the string representation of the current registe
   65: }
   66: return os<<bits; //returns string and the value returned by the generate fun
ction
   67: }
   68:
   69: FibLFSR::~FibLFSR(){  //deconstructor
   70: }
   71:
   72:
   73:
   74:
   75:
```

# PS1b: Image Encoding

## The Assignment

This assignment uses the LFSR class built in ps2a, to create a program that reads in a photo from the command line and then outputs the same image but encrypted, by having used the LFSR to encode the pixels in the image. The LFSR class was used to encode the image by left shifting all the bits in the image – thus encoding it using XOR. I was required to display the image using a SFML window and save the encrypted image to a file. The next requirement was to use the program to revert the encrypted image to its original form by rerunning the same program on it.

## Key Concepts

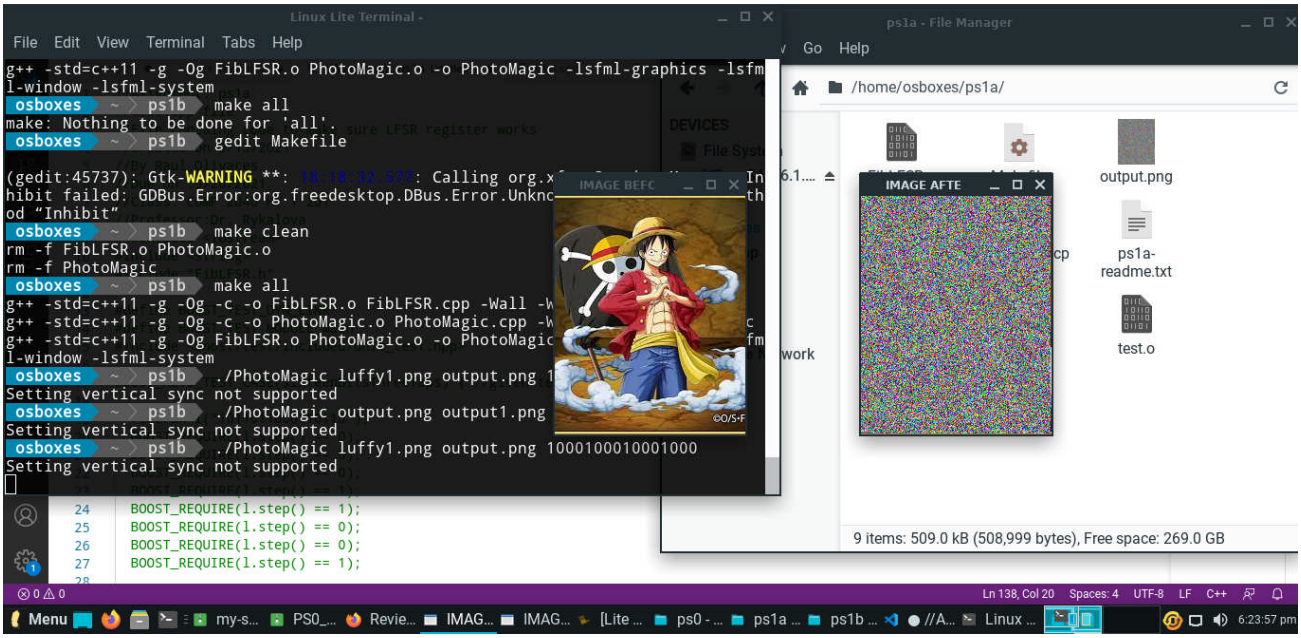
The main thing that this assignment used was the LFSR class from the previous homework, PS2a. The LFSR class that we built uses a shift register to store bits and has two methods, step and generate, that we used to left shift all the bits. We also used several SFML objects, such as textures, images and sprites to read in the file, encode the file and output the final encoded image. The PhotoMagic class was the main way we encoded the image through the use of manipulating the red, green and blue pixel using .getPixel().

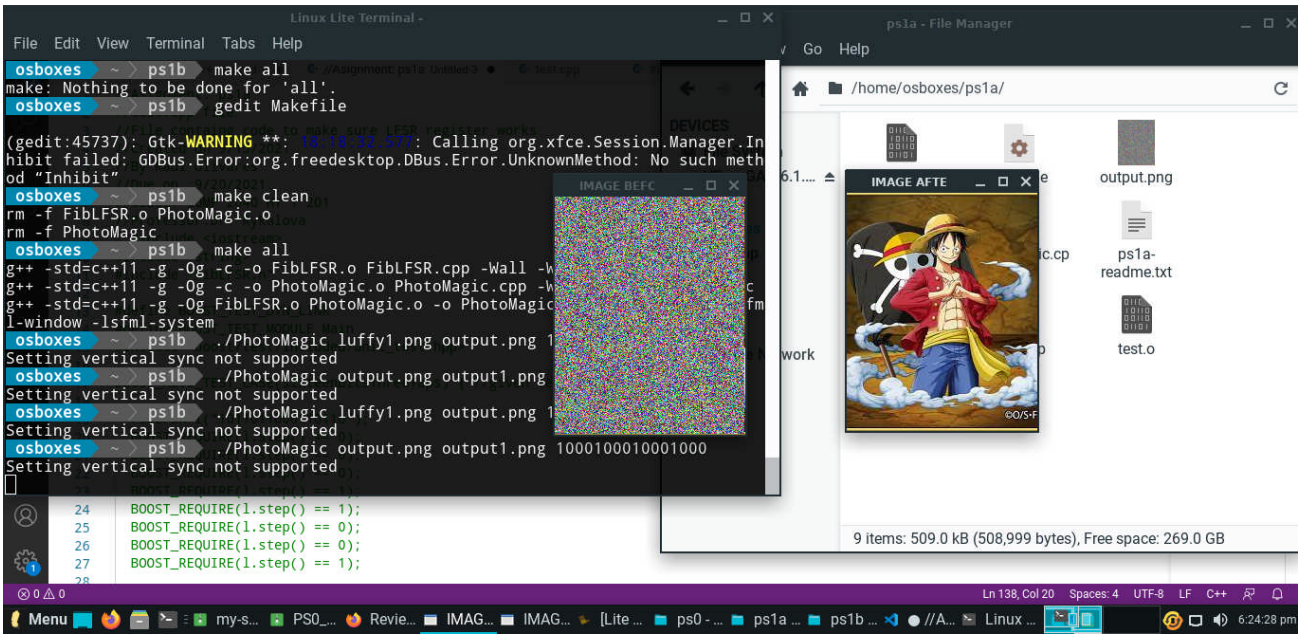
**Code for pixels from PhotoMagic.cpp**

```
pixel = image.getPixel(x,y);            //gets pixels from image

pixel.b =    pixel.b ^ Register->generate(8);     //changes blue pixels with generated code from lsfr
pixel.r =    pixel.r ^ Register->generate(8);     //changes red pixels with generated code from lsfr
pixel.g =    pixel.g ^ Register->generate(8);     //changes green pixels with generated code from lsfr
```

## What I Learned

This project taught me a couple of things such as how to use pixel formatting in c++ and how my LFSR code could be used in a real-world example. It was interesting to review previous material to create a different type of program that used most of the same code. I also learned how to pass in arguments into my code, which I wasn't too familiar with. I found the encoding portion of the assignment highly interesting because it seems to give a basic understanding of what cryptographic programming might entail. Playing around with pixels and XORing them to get an encoded image, and then displaying the final encypted image was intriguing.


Linux Lite Terminal -
File Edit View Terminal Tabs Help
g++ -std=c++11 -g -Og FibLFSR.o PhotoMagic.o -o PhotoMagic -lsfml-graphics -lsfml-window -lsfml-system
osboxes ~ ps1b make all
make: Nothing to be done for 'all'.
osboxes ~ ps1b gedit Makefile
(gedit:45737): Gtk-WARNING **: 18:18:32.577: Calling org.xfce.Session.Manager.Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.UnknownMethod: No such method "Inhibit"
osboxes ~ ps1b make clean
rm -f FibLFSR.o PhotoMagic.o
rm -f PhotoMagic
osboxes ~ ps1b make all
g++ -std=c++11 -g -Og -c -o FibLFSR.o FibLFSR.cpp -Wall -W
g++ -std=c++11 -g -Og -c -o PhotoMagic.o PhotoMagic.cpp -W
g++ -std=c++11 -g -Og FibLFSR.o PhotoMagic.o -o PhotoMagic
l-window -lsfml-system
osboxes ~ ps1b ./PhotoMagic luffy1.png output.png 1
Setting vertical sync not supported
osboxes ~ ps1b ./PhotoMagic output.png output1.png
Setting vertical sync not supported
osboxes ~ ps1b ./PhotoMagic luffy1.png output.png 1000100010001000
Setting vertical sync not supported
IMAGE BEFO
IMAGE AFTE
ps1a - File Manager
Go Help
/home/osboxes/ps1a/
output.png
ps1a-readme.txt
test.o
9 items: 509.0 kB (508,999 bytes), Free space: 269.0 GB
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);
Ln 138, Col 20 Spaces: 4 UTF-8 LF C++
Menu
6:23:57 pm


Linux Lite Terminal -
File Edit View Terminal Tabs Help
osboxes ~ ps1b make all
make: Nothing to be done for 'all'.
osboxes ~ ps1b gedit Makefile
(gedit:45737): Gtk-WARNING **: 18:18:32.577: Calling org.xfce.Session.Manager.Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.UnknownMethod: No such method "Inhibit"
osboxes ~ ps1b make clean
rm -f FibLFSR.o PhotoMagic.o
rm -f PhotoMagic
osboxes ~ ps1b make all
g++ -std=c++11 -g -Og -c -o FibLFSR.o FibLFSR.cpp -Wall -W
g++ -std=c++11 -g -Og -c -o PhotoMagic.o PhotoMagic.cpp -W
g++ -std=c++11 -g -Og FibLFSR.o PhotoMagic.o -o PhotoMagic
l-window -lsfml-system
osboxes ~ ps1b ./PhotoMagic luffy1.png output.png 1
Setting vertical sync not supported
osboxes ~ ps1b ./PhotoMagic output.png output1.png
Setting vertical sync not supported
osboxes ~ ps1b ./PhotoMagic luffy1.png output.png 1
Setting vertical sync not supported
osboxes ~ ps1b ./PhotoMagic output.png output1.png 1000100010001000
Setting vertical sync not supported
IMAGE BEFO
IMAGE AFTE
ps1a - File Manager
Go Help
/home/osboxes/ps1a/
output.png
ps1a-readme.txt
test.o
9 items: 509.0 kB (508,999 bytes), Free space: 269.0 GB
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);
Ln 138, Col 20 Spaces: 4 UTF-8 LF C++
Menu
6:24:28 pm

```
all: PhotoMagic

PhotoMagic: FibLFSR.o PhotoMagic.o
        g++ -std=c++11 -g -Og FibLFSR.o PhotoMagic.o -o PhotoMagic -lsfml-graphics -lsfml-window -lsfml-system

FibLFSR.o : FibLFSR.cpp FibLFSR.h
        g++ -std=c++11 -g -Og -c -o FibLFSR.o FibLFSR.cpp -Wall -Werror -pedantic

PhotoMagic.o: PhotoMagic.cpp
        g++ -std=c++11 -g -Og -c -o PhotoMagic.o PhotoMagic.cpp -Wall -Werror -pedantic

clean:
        rm -f FibLFSR.o PhotoMagic.o
        rm -f PhotoMagic
```

```
//Asignment: ps1b
//test.cpp file
//File containg code to make image pixels change using LFSR register.
//Created on 9/23/2021
//By Raul Olivares
//Due on :9/27/2021
//Class: COMP 2040 HY 1 201
//Professor:Dr. Rykalova

#include "FibLFSR.h"
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

void transform( sf::Image& image, FibLFSR* reg); //function declaration

int main(int argc, char* argv[])        //main function
{

    if(argc != 4)    // show user correct format to use file
    { std::cout << "Correct format to run PhotoMagic: $ ./PhotoMagic [input file] [output file] [seed]  \n";
    return -1; }

    std::string inputFile(argv[1]);
    std::string outputFile(argv[2]);    // Save the command line arguments to variables
    std::string seed(argv[3]);

FibLFSR lfsr{seed};       //initiate lfsr object

    sf::Image image;        //create image object

      if (!image.loadFromFile(inputFile))   //load image
    {return EXIT_FAILURE;}

    sf::Vector2u size = image.getSize();   //get size of the image to make appropriate window
    sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "IMAGE BEFORE LFSR");  //create first window to show input file
    sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "IMAGE AFTER LFSR"); //create second window to show outputfile

    sf::Texture texture;   //create texture object
    texture.loadFromImage(image);     //load image unto texture
    sf::Sprite sprite;     //create sprite object
    sprite.setTexture(texture);  //load texture unto sprite

transform(image,&lfsr);  //run transform function on lsfr object

sf::Texture texture2;                   // create second texture object
        texture2.loadFromImage(image);   //load encrypeted image onto texture

        sf::Sprite sprite2;              //create second sprite
        sprite2.setTexture(texture2);     //set 2nd texture onto 2nd sprite


    while(window1.isOpen() && window2.isOpen())        //while loop for open windows
    {   sf::Event event;
```

```
            while (window1.pollEvent(event))
                {if (event.type == sf::Event::Closed)   //event to close first window
                     window1.close();}

            window1.clear();                    //clear screen on first window
            window1.draw(sprite);                //draw first sprite unto first window
            window1.display();                //display window


        sf::Event event2;
        while (window2.pollEvent(event2))
           {if (event2.type == sf::Event::Closed)     //event to close second window
                window2.close();}

        window2.clear();                  //clear second window
        window2.draw(sprite2);          //draw second sprite unto second window
        window2.display();        //display second window
    }

    if (!image.saveToFile(outputFile))           //save output file with new encrypted image
        return -1;

return 0;     //exit main function
}
    void transform( sf::Image& image, FibLFSR* Register)   //transform function for pixels
{
    sf::Color pixel;            //create color pixel object



    for (int x = 0; x<200; x++)    //for loop to change all pixels in the image
    {
        for (int y = 0; y<250; y++)
        {

        pixel = image.getPixel(x,y);               //gets pixels from image

            pixel.b =    pixel.b ^ Register->generate(8);     //changes blue pixels with generated code from lsfr
            pixel.r =    pixel.r ^ Register->generate(8);      //changes red pixels with generated code from lsfr
            pixel.g =    pixel.g ^ Register->generate(8);      //changes green pixels with generated code from lsfr


        image.setPixel(x, y, pixel);   //sets new pixels unto the image
        }
    }

}

```

```
 1: //Asignment: ps1a
 2: //FibLFSR.cpp file
 3: //File containg header code for LFSR register
 4: //Created on 9/15/2021
 5: //By Raul Olivares
 6: //Due on :9/20/2021
 7: //Class: COMP 2040 HY 1 201
 8: //Professor:Dr. Rykalova
 9: #ifndef FIB_LFSR_H
10: #define FIB_LFSR_H
11:
12: #include<string>
13: #include<bitset>
14: #include<iostream>
15:
16: class FibLFSR
17: {
18:
19: public:
20:
21: FibLFSR(std::string seed); //constructor
22: ~FibLFSR();
23:
24: int step(); //step method to produce LFSR
25:
26: int generate(int k);    //generate method to produce variable containing val
ue after doubling and adding new extracted number
27:
28: friend std::ostream& operator<< (std::ostream& os, const FibLFSR lfsr); //fr
iend of ostream operations
29:
30: private:
31:
32: static const size_t ArraySize{16}; // variable used to make a set size for t
he array we need to create the register
33:
34: std::bitset<ArraySize> Register{};  //we place ArraySize in here because bit
set requires a fixed size in advance to initiate
35:
36: };
37: #endif
```

```
//Asignment: ps1a
//FibLFSR.cpp file
//File containg detailed code needed to run the LFSR register which shifts every bit
//to the left, then takes the xor value of the left most bit against 3 taps located in different part of the
//array and then returns the value, as well as placing it into the rightmost bit after the shift. Also
//performing a generate function that sets a variable initiated at 0, then performs shift operation, doubles the variable and adds
//the extracted xor value to it and then returns it.
//Created on 9/15/2021
//By Raul Olivares
//Due on :9/20/2021
//Class: COMP 2040 HY 1 201
//Professor:Dr. Rykalova
#include "FibLFSR.h"
#include<string>       //all included files needed
#include<iostream>
using namespace std; //used if we dont want to write std:: before certain codes


FibLFSR::FibLFSR(std::string seed){

    if(seed.length()>16 || seed.length()<15)  //checks the size of the seed value
   {cout<<"Seed must be 16 bits long."<<endl;}
        else{}

        for(size_t i{} ;i<seed.length();i++)
   {char bit{seed[i]};

        if(bit !='1' && bit !='0')
{cout<<"Register must use 0 & 1 binary code!!!" <<endl;}
// if to make sure only 1's and 0's were used.

Register.set(15 - i, bit =='1');  //sets the register value to hold seed value given
}
}



int FibLFSR ::step()
{
    int output = Register[15] ^ Register[13] ^ Register[12] ^ Register[10] ; //output after xoring and the taps
    Register <<= 1;   //shift operation to move all bits in register by 1 to the left
    Register.set(0, static_cast<bool>(output));   //sets the output of the xor operations into the oth place in reg
    return output;   //returns output

}

int FibLFSR:: generate(int k)
{
    uint16_t generate_output{};//initializing unsigned variable to hold value for generate input
```

```
   51:    for(int i{};i<k;i++) {//for loop
   52:     generate_output <<=1;    //shifting everything to the left
   53:     generate_output |= step();} //sets all the bits into generateoutput afte
r running step function
   54:
   55: return generate_output;  //returns generate_output
   56: }
   57:
   58: std::ostream& operator<<(std::ostream& os, const FibLFSR lfsr)
   59: {
   60: std::string bits;
   61: bits.reserve(FibLFSR::ArraySize);  //reserves space to store the bits
   62: for(int i{FibLFSR::ArraySize -1}; i>=0;i--)
   63: {
   64: bits.push_back('0'+lfsr.Register[i]);//pushes onto bits the values in the re
gister to create the string representation of the current registe
   65: }
   66: return os<<bits; //returns string and the value returned by the generate fun
ction
   67: }
   68:
   69: FibLFSR::~FibLFSR(){  //deconstructor
   70: }
   71:
   72:
   73:
   74:
   75:
```

# PS2a: N-Body Simulation

## The Assignment

For this assignment, we worked with a N-Body Simulation problem. It sets out to model the universe on a 2D plane, using Newton's laws of gravity to make the simulation realistic. We read in two command line arguments – total simulation time and the time step – and then created a stilled image of the universe on the screen. This portion of the assignment was mainly focused on reading in a file from standard I/O, and using the data found in that specific file to create sprites that would simulate the Sun, as well as a couple of planets in a scaled replica of our actual universe. This required that the corresponding distances and location of the planets in respect to the Sun were displayed with correct proportions inside of an SFML window.

## Key Concepts

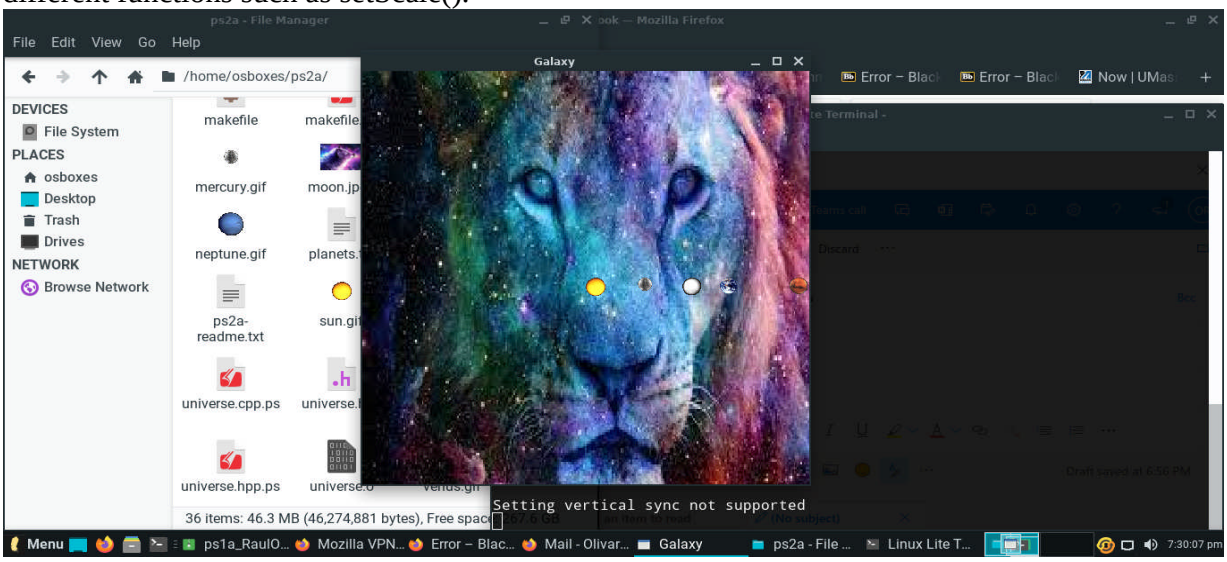
An important thing was create a method to convert X / Y positions of the planets to SFML coordinates. I was able to do this by realizing that the SFML window system sets (0, 0) as the top left corner. To convert the coordinates to the SFML system, I messed with the height and size of the given window to help create the proper coordinates. Overloading the >> operator to read in data was also important. It was also important to use the < command line operator to read in a file to standard I/O, afterwards I used cin to read the file's contents.

**Code for overloading operator**

```
std::istream& operator>> (std::istream &input, universe &cUniverse)  // Overridden operator >> for inputing from a file
```

## What I Learned

I learned about the implementation of the draw methods for the requirements of the Celestialbody class. I learned about overloading the >> operator, something that was new to me. It is something that I believe is very useful to know and will benefit me in the future. Having had to display the planets with the right coordinates helped me to learn how to convert the coordinates from the data provided to the corresponding SFML's coordinates. This helped me to learn about how the x and y coordinate coding worked in SFML. I was also able to learn about different functions such as setScale().



```
# Makefile for ps2a
# Flags to save on typing
CC= g++
CFLAGS= -Wall -Werror -ansi -pedantic
SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system

all:    NBody

# body executable
NBody:  main.o universe.o Celestialbody.o
        $(CC) main.o universe.o Celestialbody.o -o NBody $(SFMLFLAGS)

# object files
main.o: main.cpp universe.hpp
        $(CC) -c main.cpp universe.hpp Celestialbody.hpp  $(CFLAGS)

universe.o: universe.cpp Celestialbody.hpp
        $(CC) -c universe.cpp universe.hpp $(CFLAGS)

Celestialbody.o:        Celestialbody.cpp Celestialbody.hpp
        $(CC) -c Celestialbody.cpp Celestialbody.hpp $(CFLAGS)

# Cleanup
clean:
        rm *.o
        rm NBody
```

```
#include "universe.hpp"

int main(int argc, char* argv[])
{
    universe* uni = new universe();  // Create a new universe object

    std::cin >> *uni;  // Read input into the object


  sf::RenderWindow window(sf::VideoMode(windowWidth, windowHeight), "Galaxy"
);  // SFML Window

  window.setFramerateLimit(1);  // Change the framerate for future animation
window.setPosition(sf::Vector2i(400,50));

  sf::Image galaxyImage;
  if (!galaxyImage.loadFromFile("galaxy.jpg"))   // Background image
  {
    return -1;    // Quit if the file doesn't exist.
  }

  sf::Texture galaxyTexture;   // Load the image into a texture
  galaxyTexture.loadFromImage(galaxyImage);

  sf::Sprite galaxySprite;           // Load the texture into a sprite
  galaxySprite.setTexture(galaxyTexture);

  galaxySprite.setPosition(sf::Vector2f(0, 0));   // Set the position to mak
e the background look cool
galaxySprite.setScale(1.9,2.8);

  while (window.isOpen())    // Window loop
  {
    sf::Event event;    // Process events

    while(window.pollEvent(event))
    {
      if (event.type == sf::Event::Closed)   // Close window : exit
      {
        window.close();
      }

      else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
      {
        window.close();                 // Pressing escape will quit the pro
gram.
      }
    }

    window.clear();

    window.draw(galaxySprite);    // Draws galaxy background

    std::vector<Celestialbody>::iterator it;       // Display the vector of
objects

    for(it = uni->cbVector.begin(); it != uni->cbVector.end(); it++)
    {
      window.draw(*it);
    }

```

```


    window.display();
  }

  return 0;
}
```

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include "Celestialbody.hpp"
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

class universe
{
public:

  universe();

  friend std::istream& operator>> (std::istream &input, universe &cBody);
// Overridden operator >> for inputing from a file

    std::vector<Celestialbody> cbVector;

private:
    int planetNum;     // Member variables
    double universeRadius;
};
```

```
#include <vector>
#include "universe.hpp"

universe::universe()  // Default Constructor
{
  return;
}

std::istream& operator>> (std::istream &input, universe &cUniverse)  // Over
ridden operator >> for inputing from a file
{
  std::string planetNum, radius;// Get the first two numbers in the text fil
e for the amount of planets and universe radius

  input >> planetNum;// Use cin to redirect the input
  input >> radius;


  cUniverse.planetNum = atoi(planetNum.c_str()); // Now we know how many pla
nets, the radius. Convert these from string
  cUniverse.universeRadius = atof(radius.c_str());

  std::cout << "Num of planets: " << cUniverse.planetNum << std::endl;
  std::cout << "Radius: " << cUniverse.universeRadius << std::endl << std::e
ndl;

  for(int i = 0; i < cUniverse.planetNum; i++)// Loop through, create body o
bjects using the input file.
  {
    // Create a new object
    Celestialbody* cb = new Celestialbody();

    input >> *cb;  // Read input into the object

    cb->setRadius(cUniverse.universeRadius); // Set the radius and the plane
t positions.
    cb->setPosition();

    cUniverse.cbVector.push_back(*cb);    // Save the object to the vector

    std::cout << *cb;    // Test the object (debugging)
  }

  return input;
}
```

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>


const int windowHeight = 500;
const int windowWidth = 500;

class Celestialbody: public sf::Drawable
{
public:

  Celestialbody();
       // Constructors
  Celestialbody(double positionX, double positionY, double velocityX, double velocityY,
       double mass, double radius, std::string filename);

  void setRadius(float radius);
  void setPosition();              // Sets the planets positions

  friend std::istream& operator>> (std::istream &input, Celestialbody &cBody); // Overridden operator >> for inputing from a file

  friend std::ostream& operator<< (std::ostream &output, Celestialbody &cBody);  // Overriddden operator << for debugging

private:

  void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;   // Draw method

  double PositionX, PositionY;
  double VelocityX, VelocityY;
  double Mass;                        // Member variables
  double Radius;
  std::string Filename;

  sf::Image cbImage;
  sf::Sprite cbSprite;      // Image creation
  sf::Texture cbTexture;
};
```

```
#include "Celestialbody.hpp"

Celestialbody::Celestialbody()
{                           // Default Constructor
  return;
}

Celestialbody::Celestialbody(double positionX, double positionY, double velocityX, double velocityY,
            double mass, double radius, std::string filename)// Constructor
{
  PositionX = positionX;
  PositionY = positionY;  // Set member variables
  VelocityX = velocityX;
  VelocityY = velocityY;
  Mass = mass;
  Filename = filename;

  if (!cbImage.loadFromFile(Filename))   // Load the image into an image object
  {
    return;    // Quit if the file doesn't exist.
  }

  cbTexture.loadFromImage(cbImage);// Load the image into a texture
  cbSprite.setTexture(cbTexture);// Load the texture into a sprite
  cbSprite.setPosition(sf::Vector2f(PositionX, PositionY)); // Set the position from the Vector2f for position
}

void Celestialbody::setRadius(float radius)// Sets the universe radius
{
  Radius = radius;
  return;
}

void Celestialbody::setPosition()   // Sets the planets position
{
//       PositionX = (PositionX) * ((windowWidth / 2) / Radius) + (windowWidth / 2);
// PositionY = (PositionY) * ((windowHeight) / Radius) + (windowHeight / 2);
//  PositionX = ( (PositionX / Radius) * (windowWidth / 2) ) + (windowWidth / 2);
  //PositionY = ( (PositionY / Radius) * (windowHeight / 2) ) + (windowHeight / 2);
 // cbSprite.setPosition(sf::Vector2f(PositionX, PositionY)); // Set the position from the Vector2f for position
double x = (PositionX / Radius) * (windowWidth / 2);
  double y = (PositionY / Radius) * (windowHeight / 2);

  // Set the position for sprite

  cbSprite.setPosition(x + (windowWidth/2), y+(windowHeight/2));
}

void Celestialbody::draw(sf::RenderTarget& target, sf::RenderStates states) const // Drawable method
{
  target.draw(cbSprite);  // Testing outputting an image.
}

```

```
std::istream& operator>> (std::istream &input, Celestialbody &cBody)// Overr
idden operator >> for inputing from a file
{
  input >> cBody.PositionX;
  input >> cBody.PositionY;
  input >> cBody.VelocityX;    // Read input into the object
  input >> cBody.VelocityY;
  input >> cBody.Mass;
  input >> cBody.Filename;


  if (!cBody.cbImage.loadFromFile(cBody.Filename))    // Load the image into
 an image object
  {
    return input;    // Quit if the file doesn't exist.
  }

  cBody.cbTexture.loadFromImage(cBody.cbImage); // Load the image into a tex
ture
  cBody.cbSprite.setTexture(cBody.cbTexture); // Load the texture into a spr
ite
  cBody.cbSprite.setPosition(sf::Vector2f(cBody.PositionX, cBody.PositionY))
;  // Set the initial position

  return input;
}



std::ostream& operator<< (std::ostream &output, Celestialbody &cBody) // Ove
rriddden operator << for debugging
{
  output << cBody.Filename << std::endl;
  output << "Pos (x): " << cBody.PositionX << std::endl;
  output << "Pos (y): " << cBody.PositionY << std::endl;
  output << "Vel (x): " << cBody.VelocityX << std::endl;   // For debugging,
 output all the data stored in the object.
  output << "Vel (y): " << cBody.VelocityY << std::endl;
  output << "Mass: " << cBody.Mass << std::endl << std::endl;

  return output;
}
```

# PS2b: N-Body Simulation

## The Assignment

This assignment required creating a small replica of our universe that was animated using Newton's law of universal gravitation and Newton's second law of motion. It was required to have a small-scale model of the Sun and a couple planets that simulate the exact movements of their actual counterparts. It was required to have a clock that showed the elapsed time as the universe aged to the designated time that the user chose for it to stop.

## Key Concepts

The main concepts for this assignment were the Physics theories of:

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

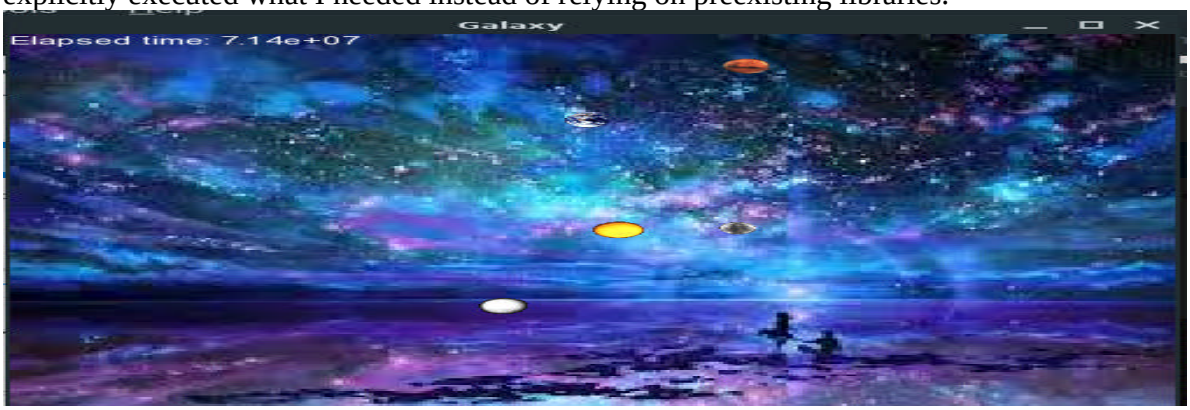
These concepts were implemented in the CelestialBody and main classes using the following formulas, such as:

```
F(Force)=M(mass)*a(acceleration)
R = square root (R2)
R2 = (Δx)^2 + (Δy)^2
Δx = x2 - x1
Δy = y2 - y1
Force = (G(Gravity) * (m1(Mass) * m2(Mass))) / r(Distance
between centers of masses)^2
```

Using these formulas, I was able to simulate the movement of the planets throughout the universe. Being able to simulate this code and combining it with with the vectors that hold the CelestialBodies required a lot of detailed precision.

## What I Learned

I found this assignment to be very intriguing and informational in combining theories with programming. I learned how to integrate physics into programming code that a computer could understand for this assignment and with that how to implement different equations in a program. I learned about how very difficult it can be to take a theory and generate code that follows its thinking. I also learned how to play music using SFML's audio library. I was able to do this by first using an internet resource to convert an mp4 file into the ogg file that SFML uses for audio to have "My universe" by Coldplay playing while the planets rotate around the Sun. I also learned how to implement a template to be able to use certain functions that my C++ compiler inside of my linux environment was having trouble compiling, so I created code that explicitly executed what I needed instead of relying on preexisting libraries.



```
# Makefile for ps2a
# Flags to save on typing
CC= g++
CFLAGS= -std=c++0x -Wall -Werror -ansi -pedantic
SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio

all:    NBody

# body executable
NBody:  main.o universe.o Celestialbody.o
        $(CC) main.o universe.o Celestialbody.o -o NBody $(SFMLFLAGS)

# object files
main.o: main.cpp universe.hpp
        $(CC) -c main.cpp universe.hpp Celestialbody.hpp  $(CFLAGS)

universe.o: universe.cpp Celestialbody.hpp
        $(CC) -c universe.cpp universe.hpp $(CFLAGS)

Celestialbody.o:        Celestialbody.cpp Celestialbody.hpp
        $(CC) -c Celestialbody.cpp Celestialbody.hpp $(CFLAGS)

# Cleanup
clean:
        rm *.o
        rm NBody
```

```
#include "universe.hpp"
#include<string.h>
#include <sstream>
#include<SFML/Audio.hpp>


template < typename T > std::string to_string( const T &n)
{                                                          //created templa
te to make to_string function because
std::ostringstream stm;                                    //because using
the regular one didnt want to compile.
stm << n;
return stm.str();
}


int main(int argc, char* argv[])
{ if(argc != 3)   // Only accepts these 3 arguments - ./ , sim time , and ti
me step
  {
    std::cout << "Usage: ./NBody [simulation time] [time step] < planets.txt
\n"; //error message
    return -1;
  }

  // Get the simulation time / time step from the command line arguments
  std::string endSimulationTime(argv[1]);
  std::string stepTime(argv[2]);

  std::cout << "Simulation time: " << endSimulationTime << "\n";
  std::cout << "Time Step: " << stepTime << "\n\n";


  double SimulationTime = 0;    // Convert these strings to doubles
  double EndSimulationTime = std::atoi(endSimulationTime.c_str());
  double TimeStep = std::atoi(stepTime.c_str());


    universe* uni = new universe();  // Create a new universe object

    std::cin >> *uni;  // Read input into the object


  sf::RenderWindow window(sf::VideoMode(windowWidth, windowHeight), "Galaxy"
);  // SFML Window

  window.setFramerateLimit(60);  // Change the framerate for animation
window.setPosition(sf::Vector2i(400,50));

  sf::Image galaxyImage;
  if (!galaxyImage.loadFromFile("galaxy.jpg"))   // Background image
  {
    return -1;    // Quit if the file doesn’t exist.
  }

    sf::Font time_font;
  time_font.loadFromFile("arial.ttf"); // load a font

  sf::Text timeText;// Text for displaying the current simulation time.


```

```
  timeText.setFont(time_font); // Select the font

  timeText.setCharacterSize(14);  // Set the character size



  sf::Music music;
  if(!music.openFromFile("galaxy.ogg"))   // Load the music file
  {
    return -1;    // error
  }

  music.play();         //play music
music.setLoop(true);    //loop music


  sf::Texture galaxyTexture;   // Load the image into a texture
  galaxyTexture.loadFromImage(galaxyImage);

  sf::Sprite galaxySprite;           // Load the texture into a sprite
  galaxySprite.setTexture(galaxyTexture);

  galaxySprite.setPosition(sf::Vector2f(0, 0));   // Set the position to make the background look cool
galaxySprite.setScale(1.9,2.8);
//galaxySprite.setScale(2.6,2.8);
std::vector<Celestialbody>::iterator y;
std::vector<Celestialbody>::iterator it;
std::vector<Celestialbody>::iterator x;

  while (window.isOpen())    // Window loop
  {
    sf::Event event;    // Process events

    while(window.pollEvent(event))
    {
      if (event.type == sf::Event::Closed)   // Close window : exit
      {
        window.close();
      }

      else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
      {
        window.close();                 // Pressing escape will quit the program.
      }
    }

    window.clear();

    window.draw(galaxySprite);    // Draws galaxy background

    timeText.setString("Elapsed time: " + to_string(SimulationTime));  // Update the time string

    window.draw(timeText);  // Display the time in the left hand corner of the window

    x = uni->cbVector.begin(); // Calculate the net force on each body object
    double force_x, force_y;
```

```


    for(int a = 0; a < uni->planetNum; a++)      // First loop goes through the whole body vector so we make sure each body object
    {                                            // gets its net force updated.
      y = uni->cbVector.begin();
      force_x = 0;
      force_y = 0;

      for(int b = 0; b < uni->planetNum; b++)      // Second loop goes through the body vector again, so that the current body object
       {                                           // gets effected by every other body object.

        if(a != b)    // Making sure a body doesn't cause a force on its own body.
        {
          force_x += find_forcex(*x, *y);
          force_y += find_forcey(*x, *y);
        }
        y++;
      }
      // Update the forces inside the current object
      x->set_forces(force_x, force_y);
      x++;
    }

    for( it = uni->cbVector.begin(); it != uni->cbVector.end(); it++)
    {                                            // Display the vector of objects
      window.draw(*it);

      it->step(TimeStep);  //move display one step

      it->setPosition();   // Update image position.
    }



    window.display();

    SimulationTime += TimeStep;      // Increase simulation time variable by the simulation step

    if(SimulationTime == EndSimulationTime)
    {                                          // Stop when we've reached the simulation time
      break;
    }
  }


  std::cout << "\n";
  for(it = uni->cbVector.begin(); it != uni->cbVector.end(); it++)
  {
    std::cout << *it << std::endl;     //prints out the final positions,velocities,etc
  }

  return 0;
```

```
165: }
```

```
 1: #include <iostream>
 2: #include <string>
 3: #include <fstream>
 4: #include <vector>
 5: #include "Celestialbody.hpp"
 6: #include <SFML/System.hpp>
 7: #include <SFML/Window.hpp>
 8: #include <SFML/Graphics.hpp>
 9: #include <SFML/Audio.hpp>
10:
11: class universe
12: {
13: public:
14:
15:   universe();
16:
17:   friend std::istream& operator>> (std::istream &input, universe &cBody);
// Overridden operator >> for inputing from a file
18:
19:     std::vector<Celestialbody> cbVector;
20: int planetNum;     // Member variables
21:     double universeRadius;
22:
23:
24: private:
25:
26: };
```

```
 1: #include <vector>
 2: #include "universe.hpp"
 3:
 4: universe::universe()  // Default Constructor
 5: {
 6:   return;
 7: }
 8:
 9:
10: std::istream& operator>> (std::istream &input, universe &cUniverse)  // Over
ridden operator >> for inputing from a file
11: {
12:   std::string planetNum, radius;// Get the first two numbers in the text fil
e for the amount of planets and universe radius
13:
14:   input >> planetNum;// Use cin to redirect the input
15:   input >> radius;
16:
17:
18:   cUniverse.planetNum = atoi(planetNum.c_str()); // Now we know how many pla
nets, the radius. Convert these from string
19:   cUniverse.universeRadius = atof(radius.c_str());
20:
21:   std::cout << "Num of planets: " << cUniverse.planetNum << std::endl;
22:   std::cout << "Radius: " << cUniverse.universeRadius << std::endl << std::e
ndl;
23:
24:   for(int i = 0; i < cUniverse.planetNum; i++)// Loop through, create body o
bjects using the input file.
25:   {
26:     // Create a new object
27:     Celestialbody* cb = new Celestialbody();
28:
29:     input >> *cb;  // Read input into the object
30:
31:     cb->setRadius(cUniverse.universeRadius); // Set the radius and the plane
t positions.
32:     cb->setPosition();
33:
34:     cUniverse.cbVector.push_back(*cb);    // Save the object to the vector
35:
36:     std::cout << *cb;    // Test the object (debugging)
37:   }
38:
39:
40:
41:   return input;
42: }
```

```
 1: #include <iostream>
 2: #include <string>
 3: #include <fstream>
 4: #include <vector>
 5: #include <SFML/System.hpp>
 6: #include <SFML/Window.hpp>
 7: #include <SFML/Graphics.hpp>
 8: #include <SFML/Audio.hpp>
 9: #include <math.h>
10:
11:
12: const int windowHeight = 500;
13: const int windowWidth = 500;
14: const double gravity = 6.67e-11;
15:
16: class Celestialbody: public sf::Drawable
17: {
18: public:
19:
20:   Celestialbody();
            // Constructors
21:   Celestialbody(double positionX, double positionY, double velocityX, double
 velocityY,
22:        double mass, double radius, std::string filename);
23:
24:   void setRadius(float radius);
25:   void setPosition();              // Sets the planets positions
26:
27:   friend double find_forcex(Celestialbody &Body1, Celestialbody &Body2);  //
 Force related methods
28:   friend double find_forcey(Celestialbody &Body1, Celestialbody &Body2);
29:   void set_forces(double forcex, double forcey);
30:
31:   void step(double time_t);    // Time step
32:
33:   friend std::istream& operator>> (std::istream &input, Celestialbody &cBody
); // Overridden operator >> for inputing from a file
34:
35:   friend std::ostream& operator<< (std::ostream &output, Celestialbody &cBod
y);  // Overriddden operator << for debugging
36:
37: private:
38:
39:   void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const
;   // Draw method
40:
41:    double _acc_x, _acc_y;
42:   double _for_x, _for_y;
43:   double PositionX, PositionY;
44:   double VelocityX, VelocityY;
45:   double Mass;                        // Member variables
46:   double Radius;
47:   std::string Filename;
48:
49:   sf::Image cbImage;
50:   sf::Sprite cbSprite;      // Image creation
51:   sf::Texture cbTexture;
52: };
```

```
#include "Celestialbody.hpp"

Celestialbody::Celestialbody()
{                           // Default Constructor
  return;
}

Celestialbody::Celestialbody(double positionX, double positionY, double velocityX, double velocityY,
            double mass, double radius, std::string filename)// Constructor
{
  PositionX = positionX;
  PositionY = positionY;  // Set member variables
  VelocityX = velocityX;
  VelocityY = velocityY;
  Mass = mass;
  Filename = filename;

  if (!cbImage.loadFromFile(Filename))   // Load the image into an image object
  {
    return;    // Quit if the file doesn't exist.
  }

  cbTexture.loadFromImage(cbImage);// Load the image into a texture
  cbSprite.setTexture(cbTexture);// Load the texture into a sprite
  cbSprite.setPosition(sf::Vector2f(PositionX, PositionY)); // Set the position from the Vector2f for position
}

void Celestialbody::setRadius(float radius)// Sets the universe radius
{
  Radius = radius;
  return;
}

void Celestialbody::setPosition()   // Sets the planets position
{

double x = (PositionX / Radius) * (windowWidth / 2);
  double y = (PositionY / Radius) * (windowHeight / 2);

  // Set the position for sprite

  cbSprite.setPosition(x + (windowWidth/2), y+(windowHeight/2));
}

void Celestialbody::draw(sf::RenderTarget& target, sf::RenderStates states) const // Drawable method
{
  target.draw(cbSprite);  // Testing outputting an image.
}


void Celestialbody::set_forces(double forcex, double forcey)
{
  _for_x = forcex;                // Sets the forces for a given object
  _for_y = forcey;
}


```

```
   58: double find_forcex(Celestialbody &Body1, Celestialbody &Body2)
   59: {                                                // Finds the force (x) bet
ween two body objects
   60:
   61:
   62:   double dx = Body2.PositionX - Body1.PositionX;    //Î\224x = x2 - x1
   63:   double dy = Body2.PositionY - Body1.PositionY;    //Î\224y = y2 - y1
   64:   double R2 = pow(dx, 2) + pow(dy, 2);
   65:   double R = sqrt(R2);
   66:   double force = (gravity * Body1.Mass * Body2.Mass) / R2;  //F = (G * M1 *
M2) / R^2
   67:   double for_x = force * (dx / R);
   68:
   69:
   70:   return for_x;
   71: }
   72:
   73:
   74: double find_forcey(Celestialbody &Body1, Celestialbody &Body2)
   75: {                                                       // Finds the force
 (y) between two body objects
   76:
   77:  double dx = Body2.PositionX - Body1.PositionX;
   78:   double dy = Body2.PositionY - Body1.PositionY;
   79:   double R2 = pow(dx, 2) + pow(dy, 2);
   80:   double R = sqrt(R2);
   81:   double force = (gravity * Body1.Mass * Body2.Mass) / R2;   //F = (G * M1 *
 M2) / R^2
   82:   double for_y = force * (dy / R);
   83:
   84:
   85:
   86:   return for_y;
   87: }
   88:
   89:
   90: void Celestialbody::step(double time_t)
   91: {
   92:
   93:   _acc_x = _for_x / Mass;  // Convert forces into acceleration (Ax = Fx / m)
   94:   _acc_y = _for_y / Mass;   //(Ay = Fy / m)
   95:
   96:
   97:   VelocityX = VelocityX + (_acc_x * time_t);      //Calculate change in velo
city (vx + Î\224t ax)
   98:   VelocityY = VelocityY - (_acc_y * time_t);      //(vy + Î\224t ay)
   99:
  100:
  101:    PositionX = PositionX + ((VelocityX) * time_t);   // Body moves based on
its velocity  (px + Î\224t vx)
  102:   PositionY = PositionY - ((VelocityY) * time_t);     //(py + Î\224t vy)
  103:
  104: }
  105:
  106: std::istream& operator>> (std::istream &input, Celestialbody &cBody)// Overr
idden operator >> for inputing from a file
  107: {
  108:   input >> cBody.PositionX;
  109:   input >> cBody.PositionY;
  110:   input >> cBody.VelocityX;    // Read input into the object
  111:   input >> cBody.VelocityY;
```

```
  input >> cBody.Mass;
  input >> cBody.Filename;


  if (!cBody.cbImage.loadFromFile(cBody.Filename))    // Load the image into
 an image object
  {
    return input;    // Quit if the file doesn’t exist.
  }

  cBody.cbTexture.loadFromImage(cBody.cbImage); // Load the image into a tex
ture
  cBody.cbSprite.setTexture(cBody.cbTexture); // Load the texture into a spr
ite
  cBody.cbSprite.setPosition(sf::Vector2f(cBody.PositionX, cBody.PositionY))
;  // Set the initial position

  cBody._for_x = 0;
  cBody._for_y = 0;        // Set force / acceleration to 0.
  cBody._acc_x = 0;
  cBody._acc_y = 0;

  return input;
}



std::ostream& operator<< (std::ostream &output, Celestialbody &cBody) // Ove
rriddden operator << for debugging
{
  output << cBody.Filename << std::endl;
  output << "Pos (x): " << cBody.PositionX << std::endl;
  output << "Pos (y): " << cBody.PositionY << std::endl;
  output << "Vel (x): " << cBody.VelocityX << std::endl;   // For debugging,
 output all the data stored in the object.
  output << "Vel (y): " << cBody.VelocityY << std::endl;
  output << "Mass: " << cBody.Mass << std::endl << std::endl;

  return output;
}
```

# PS3: Recursive Graphic

## The Assignment

This assignment, we were tasked with implementing a recursion program that was similar to the Sierpinski triangle. The main idea behind the assignment was to use recursion to create a triangle that sprouted three triangles half its size at each of its corners, that would in turn sprout another three and so on. The main program takes in two integers; the first one will determine the length of the base triangle and the second will be used to control the depth of the recursion. The program will then draw one triangle at depth 0, one base triangles with 3 child triangles at depth 1 and so on – in effect drawing triangles on top of triangles in a recursive manner.

## Key Concepts

The key concept to this program was the idea of recursion. I had to figure out a way implement the idea of triangles drawing on top of each other recursively. I was able to accomplish this by using the data structure of a tree in which I start off by implementing a base triangle that acts as the parent, which has three child triangles that will sprout from it. These child triangles will then sprout more triangles from each other in a similar pattern, in which they become the parent and so on. The length and depth of recursion will be set by the user at the start of the program.

```
                  Δ
                Base
            /     |     \
     Child Δ   Child Δ  Child Δ
     / | \     / | \    / | \
    Δ Δ Δ     Δ Δ Δ    Δ Δ Δ
   /|\/|\/|\ /|\/|\/|\ /|\/|\/|\
```

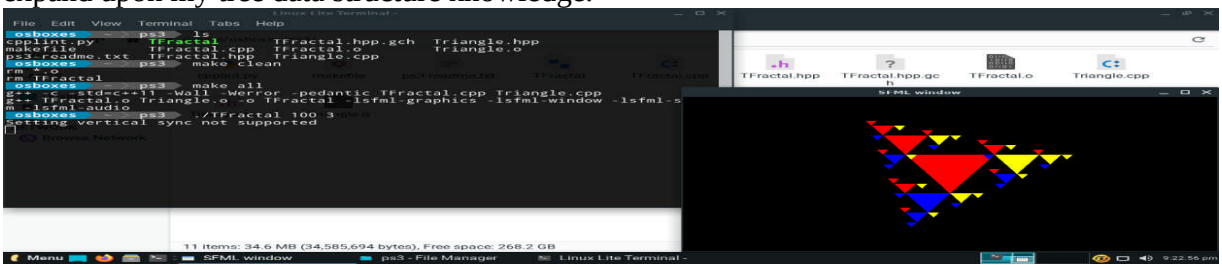
Lines 36-39 in TFractal.cpp show some of this implementation.

```
36 Triangle *triangleChild = new Triangle(baseLength/2);
36 triangleChild->addLeftChild(fTree(triangleChild,depth-1));
36 triangleChild->addRightChild(fTree(triangleChild,depth-1));
36 triangleChild->addUnderChild(fTree(triangleChild,depth-1));
```

This snippet of code shows a child triangle becoming a parent triangle that ends up having its own three child triangles.

## What I Learned

I ended up learning a lot about recursion, such as what it was and how it functioned. I got to learn some of the drawbacks to this technique and how it is important to optimize code by the way in which running higher and higher depths of the recursion would start to consume an enormous amount of memory. I also got to further my problem solving and critical thinking skills by having to figure out the ways that I could use to implement this program. I also was able to expand upon my tree data structure knowledge.



```
# Makefile for ps3
# Flags to save on typing
CC= g++
CFLAGS= -std=c++11 -Wall -Werror -pedantic
SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio

all:    TFractal

# body executable
TFractal:       TFractal.o Triangle.o
        $(CC) TFractal.o Triangle.o -o TFractal $(SFMLFLAGS)

# object files
#main.o:        main.cpp universe.hpp
#       $(CC) -c main.cpp universe.hpp Celestialbody.hpp  $(CFLAGS)
TFractal.o: TFractal.cpp Triangle.cpp
        $(CC) -c $(CFLAGS) TFractal.cpp Triangle.cpp

Triangle.o:     Triangle.cpp
        $(CC) -c $(CFLAGS) Triangle.cpp

# Cleanup
clean:
        rm *.o
        rm TFractal

```

```
//*******Main.cpp Main class file
// Created by: Raul Olivares
// On: October 19, 2021
// Assignment: PS3
// Teacher Dr. Rykalova
// Class: COMP 2040 HY 1 201
// Program: Program to draw triangles with recursion.
// Bugs: 1. Getting depth past 15.
#include <iostream>
#include <stdexcept>
#include <sstream>           //needed codes
#include <SFML/Graphics.hpp>
#include "TFractal.hpp"
#include "Triangle.hpp"


void TFractal::fTree(int depth,double length,int WindowWidth, int WindowHeight){
triangle2= new Triangle(length); //creates base triangle
triangle2->move( (WindowWidth/2) - (length/2),(WindowHeight/2)-(length/2) ); //moves triangleto appropriate position
triangle2->addLeftChild(fTree(triangle2,depth));
triangle2->addRightChild(fTree(triangle2,depth));      //creates triangle child for base triangle
triangle2->addUnderChild(fTree(triangle2,depth));

setChildPosition(triangle2);
}


//void fTree(int depth);

Triangle* TFractal::fTree(Triangle* triangle3,int depth){
if(depth<=0){           //if statement in case depth is 0
return nullptr;
}
double baseLength =triangle3->length;
//sf::Vector2f basePosition= triangle3->getPosition();
Triangle *triangleChild = new Triangle(baseLength/2);
triangleChild->addLeftChild(fTree(triangleChild,depth-1));   //triangle code for recursion
triangleChild->addRightChild(fTree(triangleChild,depth-1));
triangleChild->addUnderChild(fTree(triangleChild,depth-1));
return triangleChild;
}

void TFractal::setChildPosition(Triangle* baseTriangle){
if(baseTriangle== nullptr|| baseTriangle->triangleLeft==nullptr)
{
return;
}
sf::Vector2f basePosition{ baseTriangle->getPosition()};
double baseLength{ baseTriangle->length};
double childLength{ baseLength/2};

baseTriangle->triangleLeft->move(basePosition.x-childLength/2,basePosition.y-childLength);
baseTriangle->triangleRight->move(basePosition.x + baseLength,basePosition.y);  //moves child triangles to appropriate position
baseTriangle->triangleUnder->move(basePosition.x,basePosition.y+baseLength);

```

```
   56: setChildPosition(baseTriangle->triangleRight); //sets position
   57: setChildPosition(baseTriangle->triangleLeft);
   58: setChildPosition(baseTriangle->triangleUnder);
   59:
   60: baseTriangle->triangleUnder->setFill(sf::Color::Blue);
   61: baseTriangle->triangleRight->setFill(sf::Color::Yellow);    //creates color
triangles
   62: baseTriangle->triangleLeft->setFill(sf::Color::Red);
   63: }
   64:
   65:
   66:
   67:
   68:
   69:
   70:
   71:
   72:
   73: int main(int argc, char* argv[])
   74: { if (argc !=3)
   75: {
   76: std::cout << "Must put length of triangle[argument 1] and number of iteratio
ns[argument 2]!";
   77: return-1;}
   78:
   79: std::string length(argv[1]);  //get input variables for length and iteration
s
   80: std::string iteration(argv[2]);
   81:
   82:
   83: double Length =std::atoi(length.c_str());  //convert string to double
   84: int width{(int)Length * 6};
   85: int height{(int)Length * 4};
   86: double Iteration= std::atoi(iteration.c_str());
   87:  sf::RenderWindow window(sf::VideoMode(width, height), "SFML window");
   88:
   89:         TFractal *obj= new TFractal();     //creates Tfractal object
   90:         obj->fTree(Iteration,Length,width,height);   // runs fTree method to
 create triangles
   91:         Triangle* triangle5= obj->triangle2; //creates pointer of triangles
   92:  // Window loop
   93:   while (window.isOpen()) {
   94:     // Process events
   95:     sf::Event event;
   96:
   97:     while (window.pollEvent(event)) {
   98:       // Close window : exit
   99:       if (event.type == sf::Event::Closed) {
  100:         window.close();
  101:       } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
  102:         window.close();
  103:       }
  104:     }
  105:
  106:     window.clear();
  107:     window.draw(*triangle5);
  108:     // Call the draw object in the triangleclass
  109:     window.display();
  110:   }
  111:
  112:   return 0;
```

```
}
```

```
#ifndef TFractal_H
#define TFractal_H
#include "./Triangle.hpp"
#include <string>
#include <SFML/Graphics.hpp>

class TFractal {
public:
void fTree(int depth,double length,int WindowHeight, int WindowWidth);  //ft
ree method for creating triangles
//void fTree(int depth);
Triangle *triangle2;      //triangle pointer variable
Triangle* fTree(Triangle* triangle3, int depth);    //ftree for recursion
void setChildPosition(Triangle *triangle3);     //sets triangle children pos
ition
private:

};

#endif
```

```
#ifndef Triangle_H
#define Triangle_H
#include <string>                   //needed files
#include <SFML/Graphics.hpp>

class Triangle : public sf:: Drawable {              //class triangle inheri
ts from drawable


public:
Triangle(double);  //constructor
void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;
//draw method
 double length;  //length variable

 sf::ConvexShape triangle2; //could be private

Triangle *triangleLeft;                             //variables for child tr
iangles
Triangle *triangleRight;
Triangle *triangleUnder;
void move(int x, int y);     //move method
sf::Vector2f getPosition();          //gets position
void setFill(sf::Color);             //sets color
void addLeftChild(Triangle *triangle2);
void addRightChild(Triangle *triangle2);                    //adds child tria
ngles to base triangle
void addUnderChild(Triangle *triangle2);
//double getLength();
Triangle* getLeft();
Triangle* getRight();                //gets child triangles
Triangle* getUnder();
};


#endif
```

```
#include "Triangle.hpp"

Triangle::Triangle(double length)
{
  triangle2= sf::ConvexShape(); //creating base triangle
  triangle2.setPointCount(3);
  triangle2.setPoint(0, sf::Vector2f(0,0));
  triangle2.setPoint(1, sf::Vector2f(length,0)); //set points fortriangle
  triangle2.setPoint(2, sf::Vector2f(length / 2, length));
  triangle2.setFillColor(sf::Color::Red);
this->length = length;


return;
}


void Triangle::move(int x, int y){       //method moves triangle
triangle2.move(sf::Vector2f(x,y));
}

sf::Vector2f Triangle::getPosition(){
return triangle2.getPosition();                 //finds position of triangle
}

void Triangle::setFill(sf::Color color){
triangle2.setFillColor(color);             //adds color to triangles
}
void Triangle::addLeftChild(Triangle *triangleLefty){
        this->triangleLeft= triangleLefty;                            //cre
ates new left triangle and adds unto base triangle
}
void Triangle::addRightChild(Triangle *triangleRighty){
        this->triangleRight= triangleRighty;                       //creates
 new right triangle and adds unto base triangle

}
void Triangle::addUnderChild(Triangle *triangleUndie){                    //
creates new under triangle and adds unto base triangle
this->triangleUnder= triangleUndie;
}
Triangle* Triangle::getLeft(){   //gets left triangle
return triangleLeft;
}
Triangle* Triangle::getRight(){ //gets right triangle
return triangleRight;
}
Triangle* Triangle::getUnder(){  //gets under triangle
return triangleUnder;
}
void Triangle::draw(sf::RenderTarget& target, sf::RenderStates states) const
 // Drawable method
{
        target.draw(triangle2);
        if (this->triangleUnder != nullptr) target.draw(*triangleUnder);
        if (this->triangleRight != nullptr) target.draw(*triangleRight);
        if (this->triangleLeft != nullptr) target.draw(*triangleLeft);
  // Testing outputting an image.
}
```

# PS4a: Synthesizing a Plucked String

## The Assignment

This assignment required the implementation of a CircularBuffer, complete with Boost unit tests and exceptions. Since it will be used for the next assignment, the implementation of the CircularBuffer was the main objective of this assignment; which works by wrapping around like a circle array in order to store values. Boost unit tests were used to check the CircularBuffer for errors, and the CircularBuffer was designed to throw certain exceptions for specific errors. Some exceptions include std::invalid argument if you attempt to create a CircularBuffer with a capacity of 0 or less, and std::runtime_error if you attempt to enqueue a full CircularBuffer or dequeue or peek at an empty CircularBuffer.

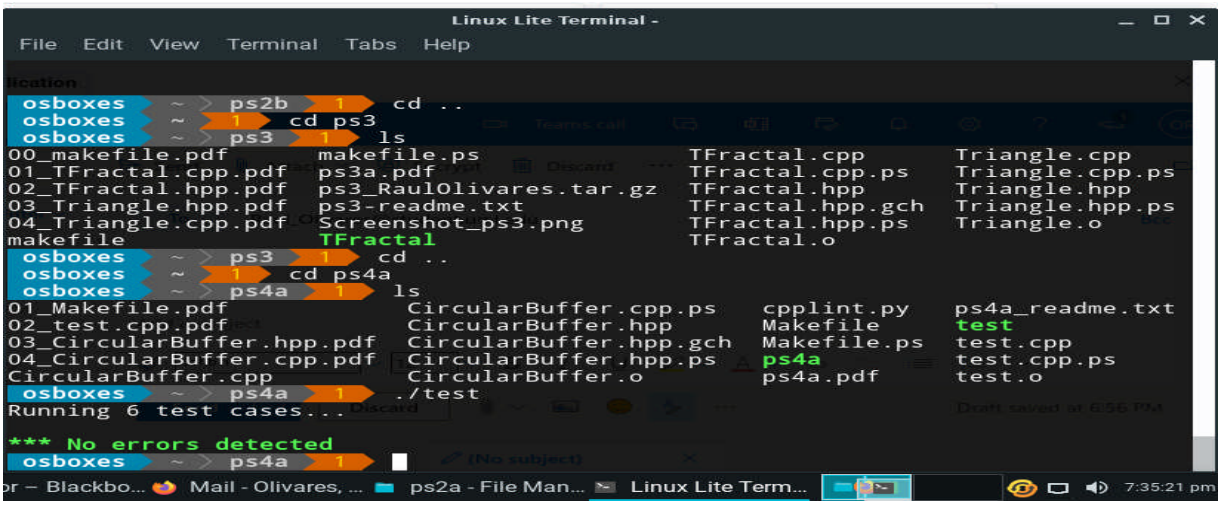
## Key Concepts

The main concepts for this assignment focused on implementing the RingBuffer using exceptions, and running unit tests. I created the CircularBuffer by using a queue as a container adapter that used a vector sequence container. In more precise words I created a queue that basically had the end and front joined so that it loops around, so that it could later function similar to a string with a musical note going through it. I used throw statements to test for invalid actions for my exceptions. Boost was also used to check the CircularBuffer and ensure that the Circularbuffer threw the correct exceptions when it was supposed to, and that no exceptions were thrown for valid actions.

```
if (isEmpty()) {
 throw
  std::runtime_error("peek: can't peek an empty ring");
}
```

## What I Learned

This assignment taught me about Exceptions and further my knowledge on the Boost library. Although having had only minimal experience with exceptions, I was able to quickly figure them out. I was also to expand my knowledge of the Boost library and get more into the habit of constantly testing my code. I also was able to expand on my knowledge of enqueuing and dequeuing techniques.



```
# Makefile for ps4a.
# Flags to save on typing.
CC= g++
CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
Boost= -lboost_unit_test_framework

# Make ps4a
all:    ps4a test

# ps4a executable
ps4a:   test.o CircularBuffer.o
        $(CC) test.o CircularBuffer.o -o ps4a $(Boost)

# test is just a basic test executable

test: test.o CircularBuffer.o
        $(CC) test.o CircularBuffer.o -o test $(Boost)

# Object files
CircularBuffer.o: CircularBuffer.cpp CircularBuffer.hpp
        $(CC) -c CircularBuffer.cpp CircularBuffer.hpp $(CFLAGS)

test.o: test.cpp CircularBuffer.hpp
        $(CC) -c test.cpp CircularBuffer.hpp $(CFLAGS)

# Cleanup object files
clean:
        rm *.o
        rm *.gch
        rm ps4a
        rm *.out
```

```
// Copyright 2021 Raul Olivares
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
#include "CircularBuffer.hpp"

BOOST_AUTO_TEST_CASE(Constructor) {  // Tests the constructor.
// No errors should be occuring.
  BOOST_REQUIRE_NO_THROW(CircularBuffer(1));
  BOOST_REQUIRE_NO_THROW(CircularBuffer(2));
  BOOST_REQUIRE_NO_THROW(CircularBuffer(3));

// Test to make sure exception and invalid_argument are getting thrown.
  BOOST_REQUIRE_THROW(CircularBuffer(-3), std::invalid_argument);
  BOOST_REQUIRE_THROW(CircularBuffer(-2), std::invalid_argument);
  BOOST_REQUIRE_THROW(CircularBuffer(-1), std::invalid_argument);
  BOOST_REQUIRE_THROW(CircularBuffer(0), std::invalid_argument);
  BOOST_REQUIRE_THROW(CircularBuffer(0), std::exception);
}

BOOST_AUTO_TEST_CASE(Size) {  // Tests the size() method.
  CircularBuffer test(3);
   test.enqueue(1);
   test.enqueue(2);
   test.enqueue(3);
  BOOST_REQUIRE(test.size() == 3);  // Should be size 3.
   test.dequeue();
   test.dequeue();
  BOOST_REQUIRE(test.size() == 1);
}

BOOST_AUTO_TEST_CASE(isEmpty) {  // Tests the isEmpty() method.
  CircularBuffer test(3);
  BOOST_REQUIRE(test.isEmpty() == true);  // Should be true.
  test.enqueue(1);
  test.enqueue(2);
  test.enqueue(3);
  BOOST_REQUIRE(test.isEmpty() == false);  // Should be false.
}

BOOST_AUTO_TEST_CASE(isFull) {  // Checks the isFull() method.
  CircularBuffer test(3);
  BOOST_REQUIRE(test.isFull() == false);  // Should be false.
  test.enqueue(1);
  test.enqueue(2);
  test.enqueue(3);
  BOOST_REQUIRE(test.isFull() == true);  // Should be true.
}

BOOST_AUTO_TEST_CASE(Enqueue) {  // Test enqueue method.
  CircularBuffer test(3);
  BOOST_REQUIRE_NO_THROW(test.enqueue(1));
  BOOST_REQUIRE_NO_THROW(test.enqueue(2));  // No errors.
  BOOST_REQUIRE_NO_THROW(test.enqueue(3));

// Throws error.
  BOOST_REQUIRE_THROW(test.enqueue(4), std::runtime_error);
}

BOOST_AUTO_TEST_CASE(Dequeue) { // Test dequeue method.
  CircularBuffer test(3);
```

```
  62:   test.enqueue(1);
  63:   test.enqueue(2);
  64:   test.enqueue(3);
  65:   BOOST_REQUIRE(test.dequeue() == 1);
  66:   BOOST_REQUIRE(test.dequeue() == 2);  // No errors.
  67:   BOOST_REQUIRE(test.dequeue() == 3);
  68:
  69: // Throws error.
  70:   BOOST_REQUIRE_THROW(test.dequeue(), std::runtime_error);
  71: }
```

```
 1: // Copyright 2021 Raul Olivares
 2: #ifndef _HOME_OSBOXES_PS4A_CIRCULARBUFFER_HPP_
 3: #define _HOME_OSBOXES_PS4A_CIRCULARBUFFER_HPP_
 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:  public:
14:   // API functions
15:
16:   explicit CircularBuffer(int capacity);  // Empty Circularbuffer
17:                                           // set at given max capacity.
18:   int size();                 // return number of items in the Circularbuffe
r.
19:   bool isEmpty();             // is size == 0?
20:   bool isFull();              // is size == capacity?
21:   void enqueue(int16_t x);    // add item x to the end.
22:   int16_t dequeue();          // delete and return item from the front
23:   int16_t peek();             // only return item from the front.
24:
25:   // Other functions
26:   void output();
27:
28:  private:
29:   std::vector<int16_t> Circularbuffer;  //vector to simulate queue
30:   int First;   //variables for queue
31:   int Last;
32:   int Capacity;
33:   int Size;
34: };
35: #endif  // _HOME_OSBOXES_PS4A_CIRCULARBUFFER_HPP_
```

```
// Copyright 2021 Raul Olivares
#include "CircularBuffer.hpp"

// Create an empty CircularBuffer with assigned capacity limit.
CircularBuffer::CircularBuffer(int capacity) {
  if (capacity < 1) {
    throw
    std::invalid_argument("CircularBuffer constructor:"
" capacity must be greater than zero");
  }

  Last = 0;
  First = 0;
  Size = 0;
  Capacity = capacity;
  Circularbuffer.resize(capacity);
  return;
}

int CircularBuffer::size() {
  return Size;  // Return number of items in the buffer.
}

bool CircularBuffer::isEmpty() {
  if (Size == 0) {  // Determine if the CircularBuffer is empty.
    return true;
  } else {
    return false;
  }
}

bool CircularBuffer::isFull() {
  if (Size == Capacity) {  // Determine if the CircularBuffer is full.
    return true;
  } else {
    return false;
  }
}

void CircularBuffer::enqueue(int16_t x) {  // add new item
  // See if the buffer is full
  if (isFull()) {
    throw
      std::runtime_error("enqueue: can’t enqueue to a full ring.");
  }

  if (Last >= Capacity) {
     Last = 0;
  }

  // If we don’t throw any exceptions, then continue on!
  Circularbuffer.at(Last) = x;

  Last++;  // Increase counter variables.
  Size++;
}

int16_t CircularBuffer::dequeue() {  // Delete and return item from the fron
t
  if (isEmpty()) {
    throw
```

```
      std::runtime_error("dequeue: can’t dequeue to an empty ring.");
  }

  int16_t previousFront = Circularbuffer.at(First);
  Circularbuffer.at(First) = 0;  // Remove from the front.

  First++;
  Size--;  // Decrease counter variables.

  if (First >= Capacity) {
    First = 0;
  }

  return previousFront;
}


int16_t CircularBuffer::peek() {  // Only return item from the front.
  if (isEmpty()) {
    throw
      std::runtime_error("peek: can’t peek an empty ring");
  }
  return Circularbuffer.at(First);
}

void CircularBuffer::output() {  // Displays the variables in stdout
  std::cout << "CircularBuffer Capacity: " << Circularbuffer.capacity() << "
\n";
  std::cout << "   First: " << First << "\n";
  std::cout << "    Last: " << Last << "\n";
  std::cout << "    Size: " << Size << "\n";
  std::cout << "\nShow the CircularBuffer \n";

  for (int x = 0; x < Capacity; x++) {
    std::cout << Circularbuffer[x] << " ";
  }
  std::cout << "\n\n";
}
```

# PS4b: StringSound implementation and SFML audio output

## The Assignment

For this assignment, I used the CircularBuffer from the previous assignment to create a model of a piano, by implementing the Karplus-Strong algorithm to simulate the plucking of a string. We were tasked with implementing a few methods to simulate the playing of a string instrument – such as pluck, tic, sample, etc. Finally, we also made the main program called KSGuitarSim; which responds to keyboard presses by the user, in which each key generated a different note that corresponded to those on a piano.

## Key Concepts

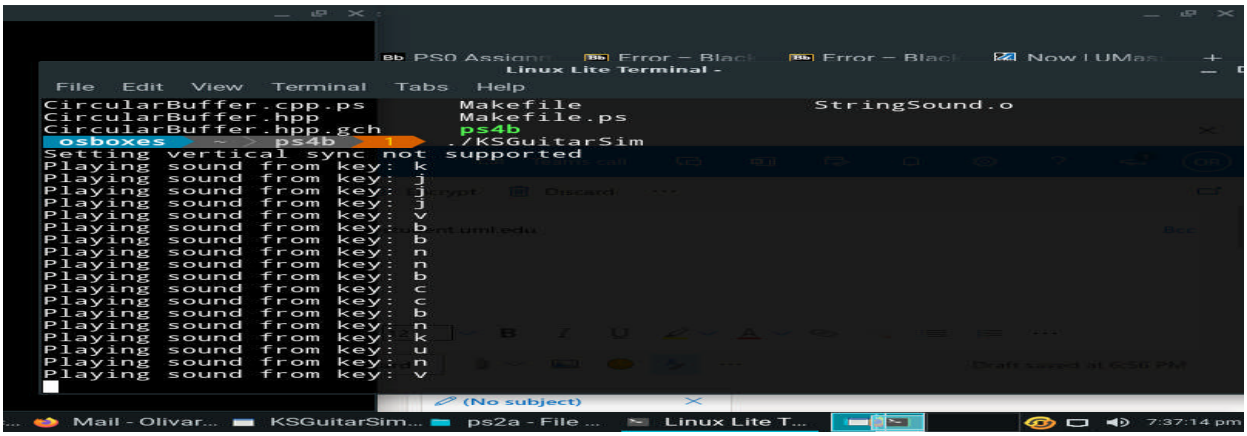
The main algorithm was used in this assignment was the Karplus-Strong algorithm, which was used to simulate the plucking of a guitar. The Karplus-Strong algorithm works by modeling frequencies, and it takes the first two values, averages them and then multiplies the result by the energy decay factor, which in our case was .996. This, along with the CircularBuffer, allowed us to model sound and created different sound notes that were similar to those generated by pressing keys on a piano.

```
int16_t avg = [&] {return (first + next) / 2;}();  // Averages first 2 values

int16_t karplus = avg * EnergyDecay;  // Multiply by EnergyDecay
// Enqueue the Karplus update.
Circularbuffer.enqueue((sf::Int16)karplus);
Tic++;
```

This code shows the processes of averaging two values, then applying the rest of the Karplus formula by applying the Energy Decay factor to get a sound note.

## What I Learned

This assignment taught me about the Karplus-Strong algorithm, which was interesting because it gave me insight on how to model sound inside of a computer program. I ran into some issues with getting the correct sound to play but was able to fix it through repeated efforts of trying out different lines of code to figure out the problem. I learned that there were shorter ways to program keyboard key functions, than using just an elongated switch statement to assign functions to various keys on the keyboard that would play the different associated piano sounds. I also learned about what Cpplint was and how to use it in my coding, although it has a lot of benefits, I do find it to be annoying in how it minimizes the freedoms of a coder to use his own unique style of coding. It also in my opinion doesn't seem to be the best template for how a code should look.



```
# Makefile for ps5b
# Flags to save on typing all this out
CC= g++
CFLAGS= -g -Wall -Werror -std=c++17 -pedantic
SFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio

# Make ps5a
all:    KSGuitarSim

# PS5B executable
KSGuitarSim:    KSGuitarSim.o StringSound.o CircularBuffer.o
        $(CC) KSGuitarSim.o StringSound.o CircularBuffer.o -o KSGuitarSim $(SFLAGS)

# Object files
KSGuitarSim.o:  KSGuitarSim.cpp StringSound.hpp
        $(CC) -c KSGuitarSim.cpp StringSound.hpp $(CFLAGS)

StringSound.o: StringSound.cpp StringSound.hpp
        $(CC) -c StringSound.cpp StringSound.hpp $(CFLAGS)

CircularBuffer.o: CircularBuffer.cpp CircularBuffer.hpp
        $(CC) -c CircularBuffer.cpp CircularBuffer.hpp $(CFLAGS)

# Cleanup object files
clean:
        rm *.o
        rm *.gch
        rm KSGuitarSim
```

```
// Copyright 2021 Raul Olivares
#ifndef _HOME_OSBOXES_PS4B_CIRCULARBUFFER_HPP_
#define _HOME_OSBOXES_PS4B_CIRCULARBUFFER_HPP_
#include <stdint.h>
#include <iostream>
#include <string>
#include <sstream>
#include <exception>
#include <stdexcept>
#include <vector>

class CircularBuffer {
 public:
  // API functions

  explicit CircularBuffer(int capacity);  // Empty Circularbuffer
                                          // set at given max capacity.
  int size();                 // return number of items in the Circularbuffe
r.
  bool isEmpty();             // is size == 0?
  bool isFull();              // is size == capacity?
  void enqueue(int16_t x);    // add item x to the end.
  int16_t dequeue();          // delete and return item from the front
  int16_t peek();             // only return item from the front.

  // Other functions
  void output();

 private:
  std::vector<int16_t> Circularbuffer;  // vector to simulate queue
  int First;   // variables for queue
  int Last;
  int Capacity;
  int Size;
};
#endif  // _HOME_OSBOXES_PS4B_CIRCULARBUFFER_HPP_"
```

```
// Copyright 2021 Raul Olivares
#ifndef _HOME_OSBOXES_PS4B_STRINGSOUND_HPP_
#define _HOME_OSBOXES_PS4B_STRINGSOUND_HPP_
#include <cmath>
#include <string>
#include <vector>
#include <iostream>
#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include "CircularBuffer.hpp"

const int SampleRate = 44100;
const double EnergyDecay = 0.996;

class StringSound {
 public:
  // create a StringSound of the given freq using a rate of 44,100
  explicit StringSound(double frequency);

  // create a StringSound with size and initial values of the vector init
  explicit StringSound(std::vector<sf::Int16> init);

  // pluck the StringSound by replacing the buffer with random valuess
  void pluck();
  void tic();              // advance simulation one time step
  sf::Int16 sample();      // return the current sample
  int time();              // returns number of times tic has been called
 private:
  CircularBuffer Circularbuffer;
  int Ndisplacement;
  int Tic;
};
#endif  // _HOME_OSBOXES_PS4B_STRINGSOUND_HPP_
```

```
// Copyright 2021 Raul Olivares
#include "StringSound.hpp"
#include <vector>
#include <random>

  // Create a guitar string of the given freq using a rate of 44,100
StringSound::StringSound(double frequency):

  Circularbuffer(ceil(SampleRate / frequency)) {
  // CircularBuffer will be this large.
  if (frequency <=0) {
  throw  // exception handling
  std::invalid_argument("frequency can’t be zero or negative");}
  Ndisplacement = ceil(SampleRate / frequency);

  // Enqueue N (44,100 / freq) 0’s.
  for (int i = 0; i < Ndisplacement; i++) {
    Circularbuffer.enqueue((int16_t)0);
  }
  // Set initial starting value for Tic.
  Tic = 0;
}

// Create a guitar string with size and initial values of the StringVector
StringSound::StringSound(std::vector<sf::Int16> StringVector):
  Circularbuffer(StringVector.size()) {
  // CircularBuffer will be as large as the array.
  Ndisplacement = StringVector.size();

  // Iterator to keep track of the vector.
  std::vector<sf::Int16>::iterator it;

  // Enqueue all the items in the vector.
  for (it = StringVector.begin(); it < StringVector.end(); it++) {
    Circularbuffer.enqueue((int16_t)*it);
  }
  // Set initial starting value for Tic.
  Tic = 0;
}


  // pluck the string by replacing the buffer with random values
void StringSound::pluck() {
  // Remove Ndisplacement items
  for (int i = 0; i < Ndisplacement; i++) {
    Circularbuffer.dequeue();
  }
// Add Ndisplacement random items between -32768 to 32767
  for (int i = 0; i < Ndisplacement; i++) {
  std::random_device rseed;
  std::mt19937 rng(rseed());       // used to create random number
  std::uniform_int_distribution<int>
  dist((0-32768), 32767);

    Circularbuffer.enqueue((sf::Int16)(dist(rng) & 0xffff));
  }

  return;
}

  // advance simulation one time step
```

```
void StringSound::tic() {
  // First get the first value, and dequeue it at the same time.
  int16_t first = Circularbuffer.dequeue();

  // Get the next value by peek()
  int16_t next = Circularbuffer.peek();

  // Apply the Karplus formula
int16_t avg = [&] {return (first + next) / 2;}();  // Averages first 2 value
s

  int16_t karplus = avg * EnergyDecay;  // Multiply by EnergyDecay
  // Enqueue the Karplus update.
  Circularbuffer.enqueue((sf::Int16)karplus);
  Tic++;

  return;
}

  // return current sample
sf::Int16 StringSound::sample() {
  // Get the value of the item at the front of the CircularBuffer
  sf::Int16 sample = (sf::Int16)Circularbuffer.peek();

  return sample;
}

  // returns the number of times tic has been called
int StringSound::time() {
  return Tic;
}
```

```
// Copyright 2021 Raul Olivares
#ifndef _HOME_OSBOXES_PS4B_CIRCULARBUFFER_HPP_
#define _HOME_OSBOXES_PS4B_CIRCULARBUFFER_HPP_
#include <stdint.h>
#include <iostream>
#include <string>
#include <sstream>
#include <exception>
#include <stdexcept>
#include <vector>

class CircularBuffer {
 public:
  // API functions

  explicit CircularBuffer(int capacity);  // Empty Circularbuffer
                                          // set at given max capacity.
  int size();                 // return number of items in the Circularbuffe
r.
  bool isEmpty();             // is size == 0?
  bool isFull();              // is size == capacity?
  void enqueue(int16_t x);    // add item x to the end.
  int16_t dequeue();          // delete and return item from the front
  int16_t peek();             // only return item from the front.

  // Other functions
  void output();

 private:
  std::vector<int16_t> Circularbuffer;  // vector to simulate queue
  int First;   // variables for queue
  int Last;
  int Capacity;
  int Size;
};
#endif  // _HOME_OSBOXES_PS4B_CIRCULARBUFFER_HPP_"
```

```
// Copyright 2021 Raul Olivares
#include "CircularBuffer.hpp"

// Create an empty CircularBuffer with assigned capacity limit.
CircularBuffer::CircularBuffer(int capacity) {
  if (capacity < 1) {
    throw
    std::invalid_argument("CircularBuffer constructor:"
" capacity must be greater than zero");
  }

  Last = 0;
  First = 0;
  Size = 0;
  Capacity = capacity;
  Circularbuffer.resize(capacity);
  return;
}

int CircularBuffer::size() {
  return Size;  // Return number of items in the buffer.
}

bool CircularBuffer::isEmpty() {
  if (Size == 0) {  // Determine if the CircularBuffer is empty.
    return true;
  } else {
    return false;
  }
}

bool CircularBuffer::isFull() {
  if (Size == Capacity) {  // Determine if the CircularBuffer is full.
    return true;
  } else {
    return false;
  }
}

void CircularBuffer::enqueue(int16_t x) {  // add new item
  // See if the buffer is full
  if (isFull()) {
    throw
      std::runtime_error("enqueue: can’t enqueue to a full ring.");
  }

  if (Last >= Capacity) {
     Last = 0;
  }

  // If we don’t throw any exceptions, then continue on!
  Circularbuffer.at(Last) = x;

  Last++;  // Increase counter variables.
  Size++;
}

int16_t CircularBuffer::dequeue() {  // Delete and return item from the fron
t
  if (isEmpty()) {
    throw
```

```
      std::runtime_error("dequeue: can’t dequeue to an empty ring.");
  }

  int16_t previousFront = Circularbuffer.at(First);
  Circularbuffer.at(First) = 0;  // Remove from the front.

  First++;
  Size--;  // Decrease counter variables.

  if (First >= Capacity) {
    First = 0;
  }

  return previousFront;
}


int16_t CircularBuffer::peek() {  // Only return item from the front.
  if (isEmpty()) {
    throw
      std::runtime_error("peek: can’t peek an empty ring");
  }
  return Circularbuffer.at(First);
}

void CircularBuffer::output() {  // Displays the variables in stdout
  std::cout << "CircularBuffer Capacity: " << Circularbuffer.capacity() << "
\n";
  std::cout << "   First: " << First << "\n";
  std::cout << "    Last: " << Last << "\n";
  std::cout << "    Size: " << Size << "\n";
  std::cout << "\nShow the CircularBuffer \n";

  for (int x = 0; x < Capacity; x++) {
    std::cout << Circularbuffer[x] << " ";
  }
  std::cout << "\n\n";
}
```

# PS5: Edit Distance

## The Assignment

This assignment asked me to implement a program that would find the optimal alignment of two strings. It was asked of me to use the Needleman-Wunsch method to calculate the most optimal method to align two strings together. I was asked to use the most optimal idea for this program and decided upon dynamic programming, so that this program calculated different edit distances by finding the most efficient possible way of traversing a matrix by taking in the different calculations of traversing through it and finding the most optimal calculation for its traversal.

## Key Concepts

One of the most important concepts that was needed for this program is known as the Needleman-Wunsch method, which is a way of using dynamic programming to calculate subproblems, and then use those subproblems to find the most efficient solution. In the case of this specific program, we used an NxM matrix to do so. This was created by first calculating the different edit distances – and then using those solutions to find the next round of edit distances, until you've arrived at the solution in the [0][0] cell of the matrix.

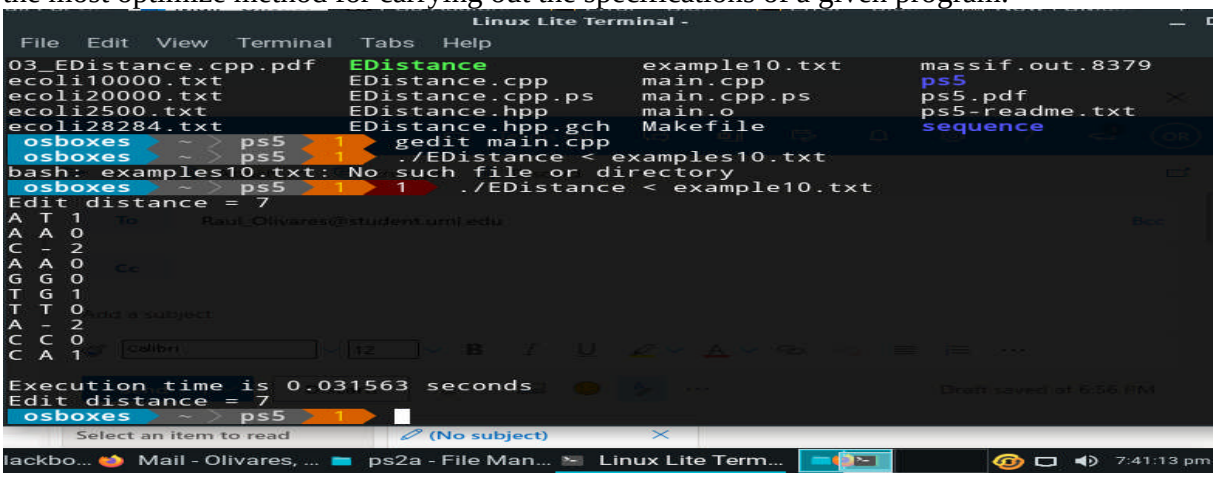
We were able to recover the least time consuming path that our algorithm could take by retracing the different steps that could be taken through the matrix. We did this by using the minimum of these.

opt[i][j] = min { opt[i+1][j+1] + 0/1, opt[i+1][j] + 2, opt[i][j+1] + 2 }

Using these three methods throughout the matrix, I was able to find the most efficient way to get to the top left most cell of the matrix ([0][0]), approaching from the right most bottom part through the most efficient edit distance, and then being able to trace those steps back to back.

## What I Learned

I learned a few things from this assignment. First, that recursion would not have been even half as an efficient method to implement this program as dynamic programming. I also learned about valgrind and how to utilize it to be able to determine the memory usage on my machine from running certain programs. I also found a nice way of displaying the results of memory usage on my operating system from running my program and displaying those results using a massif visualizer. Also that using the Needleman-Wunsch method to calculate subproblems was pretty interesting and helped in creating the most efficient method for running through this type of program, so it has given me great insight into how a programmer must think of all the possible outcomes and memory consumptions of a program and be able to determine the most optimize method for carrying out the specifications of a given program.



```
# Makefile for ps4
# Flags to save on typing all this out
CC= g++
CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
SFLAGS= -lsfml-system

# Make ps5
all:    EDistance

# body executable
EDistance:      main.o EDistance.o
        $(CC) main.o EDistance.o -o EDistance $(SFLAGS)

# object files
main.o: main.cpp EDistance.hpp
        $(CC) -c main.cpp EDistance.hpp $(CFLAGS)

EDistance.o:    EDistance.cpp EDistance.hpp
        $(CC) -c EDistance.cpp EDistance.hpp $(CFLAGS)

# Cleanup
clean:
        rm *.o
        rm *.gch
        rm EDistance
```

```
#include "EDistance.hpp"

int main(int argc, const char* argv[])
{
  // Time calculations
  sf::Clock clock;
  sf::Time t;

  // Read in two strings from stdin
  std::string string1, string2;
  std::cin >> string1 >> string2;

  // Declare a EDistance object
  EDistance edistance_test(string1, string2);

  // Find the Edit Distance
  int distance = edistance_test.OptDistance();

  // Get the string alignment
  std::string alignment = edistance_test.Alignment();

  std::cout << "Edit distance = " << distance << "\n";
  std::cout << alignment;  // Print out the edit distance

  t = clock.getElapsedTime();
  std::cout << "\nExecution time is " << t.asSeconds() << " seconds \n";
  // cant see on my vm without this
  std::cout << "Edit distance = " << distance << "\n";

  return 0;
}
```

```
#ifndef EDistance_HPP
#define EDistance_HPP

#include <iostream>
#include <iomanip>
#include <sstream>
#include <string>
#include <stdexcept>
#include <vector>
#include <math.h>
#include <SFML/System.hpp>

class EDistance
{
  public:
    EDistance();
    EDistance(std::string string1, std::string string2);
    ~EDistance();
    int penalty(char a, char b);
    int min(int a, int b, int c);
    int OptDistance();
    std::string Alignment();

  private:
    std::string _string1, _string2;

    std::vector< std::vector<int> > matrix;  //vector of int vectors
};

#endif
```

```
#include "EDistance.hpp"
EDistance::EDistance()
{  // Default constructor
}

EDistance::EDistance(std::string string1, std::string string2)
{// Contructor with parameters
  _string1 = string1;
  _string2 = string2;
}

EDistance::~EDistance()  // Destructor
{
}

int EDistance::penalty(char a, char b)
{                   // Returns the penalty of the two characters.
  if(a == b)
  {  // Equal characters
    return 0;
  }

  else if(a != b)
  {  // not equal and no spaces
    return 1;
  }
  else  // If something fails, return a -1.
  return -1;  // We can check this for errors.
}


// Finds the minimum integer
int EDistance::min(int a, int b, int c)
{
  if(a < b && a < c)
  {
    return a;
  }

  else if(b < a && b < c)
  {
    return b;
  }

  else if(c < a && c < b)
  {
    return c;
  }
  else { /* // They are all equal, return a random one
int random;
auto ret = a;
ret=0;
random = (rand()%3);
if(random == 0){          //dont know why using this adds numbers to best al
lignment
ret = a;}
else if(random == 1){
ret = b;}
else if(random == 2){
ret = c;}
    return ret;}*/
```

```
auto equal = [&](){return a;};
return equal();}
}

// Finds the optimal distance between the two strings
int EDistance::OptDistance()
{

  int i, j;
  int N = _string1.length();
  int M = _string2.length();

  for(i = 0; i <= M; i++)
  {
    std::vector<int> tmp;
    matrix.push_back(tmp);

    for(j = 0; j <= N; j++)
    {
      matrix.at(i).push_back(0);
    }
  }

  // Start filling out the bottom row
  for(i = 0; i <= M; i++)
  {
    // Very bottom row from left to right
    matrix[i][N] = 2 * (M - i);
  }

  // fills out the side row.
  for(j = 0; j <= N; j++)
  {
    // Very right most column from top to bottom
    matrix[M][j] = 2 * (N - j);
  }

  // calculate the inner sub problems in the matrix
  for(i = M - 1; i >= 0; i--)
  {
    for(j = N - 1; j >= 0; j--)
    {
         //use of the penalty method for first optimization variable
      int opt1 = matrix[i+1][j+1] + penalty(_string1[j], _string2[i]);
      int opt2 = matrix[i+1][j] + 2;
      int opt3 = matrix[i][j+1] + 2;
    //min of 3 numbers
      matrix[i][j] = min(opt1, opt2, opt3);
    }
  }

  return matrix[0][0];
}


std::string EDistance::Alignment()  // Returns the alignment
{
  // Let’s declare a stringstream object to hold the string we want to retur
n.
  std::ostringstream returnString;

```

```
  int M = _string2.length();    // Get M(rows) & N(columns) for going throug
h the Matrix
  int N = _string1.length();
  int i = 0, j = 0;
  int Penalty, opt1, opt2, opt3;
  std::string FinalReturnString;

  while(i < M || j < N) // Need to run until we reach the far bottom right c
orner
  {
    try{  //try and catch blocks for out of range exceptions
       Penalty =  penalty(_string1[j], _string2[i]);
      opt1 = matrix.at(i+1).at(j+1) + Penalty;
    }
    catch(const std::out_of_range& error)
    {
      opt1 = -1;
    }
    try{
       opt2 = matrix.at(i+1).at(j) + 2;
    }catch(const std::out_of_range& error)
    {
      opt2 = -1;
    }
    try{
     opt3 = matrix.at(i).at(j+1) + 2;
    }catch(const std::out_of_range& error)
    {
      opt3 = -1;
    }

    if(matrix[i][j] == opt1)  // Move diagonally
    {
      returnString << _string1[j] << " " <<  _string2[i] << " "  << Penalty
<< "\n";
      i++;
      j++;
    }

    else if(matrix[i][j] == opt3)  // Move right
    {
      returnString << _string1[j] << " -" << " 2\n";
      j++;
    }

    else if(matrix[i][j] == opt2)  // Move down
    {
      returnString << "- " << _string2[i] << " 2\n";
      i++;
    }
  }

  // Get the string from the ostringstream object & return it.
  FinalReturnString = returnString.str();
  return FinalReturnString;
}
```

# PS6: RandWriter

## The Assignment

This assignment required a class that implements a Markov chain to model English text. The program was designed to take an input text, and output pseudo-random text using the RandWriter class.

Markov chains are statistical models of text, which count the occurrences and sequences of characters in an English word, sentence, paragraph, or even longer text documents. The RandWriter class uses several methods, such as 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 RandWriterclass to generate a psudo-random text based on the given text input given to the program by using the generate() method, which returns a string of random characters following a given k-gram to create it. I was also asked to create a Boost testing file to test some of the required functions.

## Key Concepts

The key idea behind this assignment is to understand and implement Markov chains, which are statistical models of English text. To build this Markov Model class, I used a map data structure, which I set up 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 occurences of the given k-gram. The map structure helped to be able to search through the map to find out the number of instances of the kgram using an iterator.

RandWriter.hpp code for map

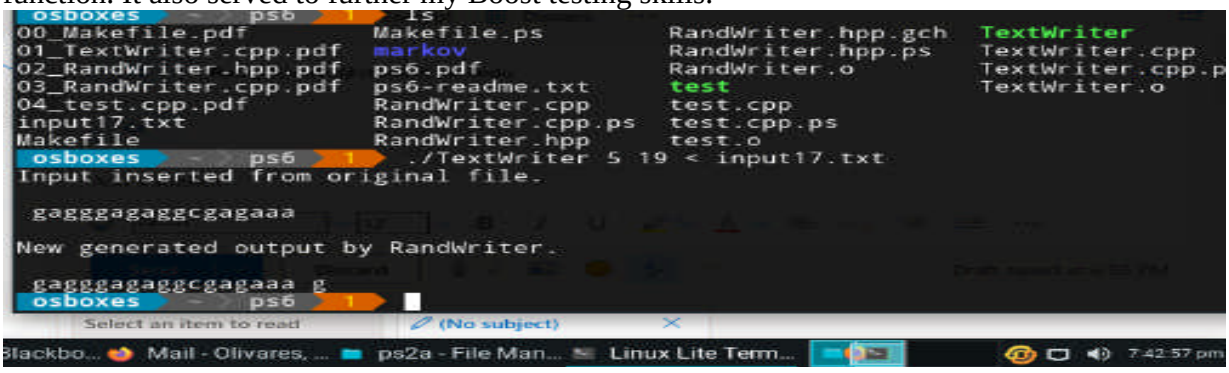
```
std::map <std::string, int> kgramsMap;
```

RandWriter.cpp code for iterator search

```
std::map<std::string, int>::iterator it;

it = kgramsMap.find(k_gram);  // search if given kgram is in map

if (it == kgramsMap.end()) {
 throw  // Throw an exception if not contained
  std::runtime_error("Error - Could not find the given k_gram!");
}
```

## What I Learned

This assignment taught me about Markov chains, what they were and how to model them with C++ programming language. It gave me an insight into how some common used programs such as auto text are made. It helped me to develop my knowledge of maps and build up my skills with using them. It also helped me to learn about the different ways of implementing random numbers in C++, since I was given a requirement to not use the standard srand() function. It also served to further my Boost testing skills.



```
# Makefile for ps6
# Flags to save on typing
CC = g++
CFLAGS = -g -Wall -Werror -std=c++17 -pedantic
SFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
Boost = -lboost_unit_test_framework

# Make ps6 and Boost test
all:    TextWriter test

# PS6 executable
TextWriter:     TextWriter.o RandWriter.o
        $(CC) TextWriter.o RandWriter.o -o TextWriter

test:   test.o RandWriter.o
        $(CC) test.o RandWriter.o -o test $(Boost)

# Object files
TextWriter.o:TextWriter.cpp RandWriter.hpp
        $(CC) -c TextWriter.cpp RandWriter.hpp $(CFLAGS)

RandWriter.o:RandWriter.cpp RandWriter.hpp
        $(CC) -c RandWriter.cpp  RandWriter.hpp $(CFLAGS)

test.o:test.cpp
        $(CC) -c test.cpp $(Boost)

# Cleanup object files
clean:
        rm *.o
        rm *.gch
        rm TextWriter
        rm test
```

```
// Copyright 2021 Raul Olivares
#include <string>
#include "RandWriter.hpp"

int main(int argc, const char* argv[]) {
  if (argc != 3) {
    std::cout << "Correct Input is ./TextWriter (int K) (int L)\n";
    return 0;
  }

  std::string stringK(argv[1]);  // strings made from arguments
  std::string stringL(argv[2]);
  int k = std::stoi(stringK);  // turn strings into ints
  int L = std::stoi(stringL);
  std::string input = "";
  std::string currentTextFile = "";

  // Reads file into currenText then transfers to input
  while (std::cin >> currentTextFile) {
    input += " " + currentTextFile;
    currentTextFile = "";
  }

  std::cout << "Input inserted from original file.\n\n";

  // Shows L amount of characters from original File
  for (int a = 0; a < (L+1); a++) {
    std::cout << input[a];
  }

  std::string outputString = "";
  RandWriter textWriter(input, k);
  outputString += "" + textWriter.generate(input.substr(0, k), (L+1));

  // Show generated output
  std::cout << "\n\nNew generated output by RandWriter.\n\n";

  for (int a = 0; a < (L+1); a++) {
    std::cout << outputString[a];
  }

  std::cout << "\n";

  return 0;
}
```

```
// Copyright 2021 Raul Olivares
#ifndef _HOME_OSBOXES_PS6_RANDWRITER_HPP_
#define _HOME_OSBOXES_PS6_RANDWRITER_HPP_
#include <algorithm>
#include <iostream>
#include <map>
#include <string>
#include <stdexcept>


class RandWriter {
 public:
  RandWriter(std::string text, int k);  // constructor
  int order_k();

  int freq(std::string k_gram);

  int freq(std::string k_gram, char c);

  char k_Rand(std::string k_gram);

  std::string generate(std::string k_gram, int L);

  friend std::ostream& operator<< (std::ostream &out, RandWriter &mm);

 private:
  int order;
  std::map <std::string, int> kgramsMap;
  std::string alphabet;
};
#endif  // _HOME_OSBOXES_PS6_RANDWRITER_HPP_
```

```
 1: // Copyright 2021 Raul Olivares
 2: #include "RandWriter.hpp"
 3: #include <algorithm>
 4: #include <map>
 5: #include <string>
 6: #include <stdexcept>
 7: #include <vector>
 8: #include <utility>
 9:
10:   // Creates a Markov model of order k from the given text.
11: RandWriter::RandWriter(std::string text, int k) {
12:   order = k;  // set the order
13:
14:   // Seed the random number generator
15:   srand((int)time(NULL)); //NOLINT
16:
17:   std::string CircularText = text;
18:
19:   for (int a = 0; a < order; a++) {  // Makes text wrap around
20:     CircularText.push_back(text[a]);
21:   }
22:
23:   int textLength = text.length();  // Find the text’s length
24:
25:
26:   char tmp;
27:   bool inAlphabet = false;  // set the alphabet.
28:
29:   for (int i = 0; i < textLength; i++) {
30:   tmp = text.at(i);  // Go through text and pick out
31:     inAlphabet = false;  //  all the individual letters
32:
33:     // See if this letter has been added to the alphabet
34:     for (unsigned int y = 0; y < alphabet.length(); y++) {
35:       if (alphabet.at(y) == tmp) {
36:         inAlphabet = true;   // Match it as being in the alphabet.
37:       }
38:     }
39:
40:     if (!inAlphabet) {  // push back to the alphabet if not in alphabet
41:       alphabet.push_back(tmp);
42:     }
43:   }
44:
45:   std::sort(alphabet.begin(), alphabet.end());
46:
47:   std::string tmpString;
48:   int x, y;
49:
50:   // Do up to text.length() substring comparisons.
51:   // This first part just "finds" kgrams and puts a "0" next to them.
52:   for (x = order; x <= order + 1; x++) {
53:     // Go through the entire text.
54:     for (y = 0; y < textLength; y++) {
55:       // This collects all given kgrams, and adds a "0" to increment later
56:       tmpString.clear();
57:       tmpString = CircularText.substr(y, x);  // current kgram we want.
58:
59:       // Insert the 0.
60:       kgramsMap.insert(std::pair<std::string, int>(tmpString, 0));
61:     }
```

```
  }

  // Need an iterator for going through the kgrams map.
  std::map<std::string, int>::iterator it;
  int countTmp = 0;

  for (x = order; x <= order + 1; x++) {
    // Go through the entire text.
    for (y = 0; y < textLength; y++) {
      // Let’s get the current kgram we’re comparing against.

      tmpString.clear();
      tmpString = CircularText.substr(y, x);

      it = kgramsMap.find(tmpString);  // kgram’s current count.
      countTmp = it->second;
      countTmp++;

      kgramsMap[tmpString] = countTmp;  // Reinsert the count into the map.
    }
  }
}

int RandWriter::order_k() {
  return order;
}  // Order k of Markov model


// Frequency of kgram in text.
int RandWriter::freq(std::string k_gram) {
  if (k_gram.length() != (unsigned)order) {
  throw   // Throw an exception if kgram is not of length k
      std::runtime_error("3Error - k_gram not of length k.");
  }

  std::map<std::string, int>::iterator it;
  it = kgramsMap.find(k_gram);  // Use std::map::find to see
                                // if we can find the kgram.
  if (it == kgramsMap.end()) {
    return 0;
  }  // If it equals map::end, it wasnt contained in map

  return it->second;  // return the given kgram
}


int RandWriter::freq(std::string k_gram, char c) {
  if (k_gram.length() != (unsigned)order) {
    throw  // Throw an exception if kgram is not of length k
      std::runtime_error("2Error - k_gram not of length k.");
  }

  // use std::map::find to see if we can find the kgram + c.
  std::map<std::string, int>::iterator it;
  k_gram.push_back(c);
  it = kgramsMap.find(k_gram);

  if (it == kgramsMap.end()) {
    return 0;  // If it equals map::end, it wasnt contained in map
  }

```

```
  return it->second;  // return the given kgram
}

// Returns a random character following the given kgram
char RandWriter::k_Rand(std::string k_gram) {
  if (k_gram.length() != (unsigned)order) {
    throw  // Throw an exception if kgram is not of length k.
      std::runtime_error("1Error - k_gram not of length k (randk)");
  }

  // iterator for going through the kgrams map.
  std::map<std::string, int>::iterator it;

  it = kgramsMap.find(k_gram);  // search if given kgram is in map

  if (it == kgramsMap.end()) {
    throw  // Throw an exception if not contained
      std::runtime_error("Error - Could not find the given k_gram!");
  }

  // Get the freq of the given kgram
  int kgram_freq = freq(k_gram);
  int random_value = rand() % kgram_freq; //NOLINT
  double test_freq = 0;
  auto createRandy = [=] () {
  return static_cast<double>(random_value) / kgram_freq; };
  double randomNumber = createRandy();
  double last_values = 0;


  // Go through all the letters.
  for (unsigned int a = 0; a < alphabet.length(); a++) {
    // calculates the probability
    test_freq =  static_cast<double>(freq(k_gram, alphabet[a])) / kgram_freq
;

    if (randomNumber < test_freq + last_values && test_freq != 0) {
      return alphabet[a];
    }

    last_values += test_freq;
  }

  return 0;
}

std::string RandWriter::generate(std::string k_gram, int L) {
  if (k_gram.length() != (unsigned)order) {
    throw  // Throw an exception if kgram is not of length k.
      std::runtime_error("Error - k_gram not of length k. (gen)");
  }

  std::string final_string = "";
  char return_char;

  final_string += "" + k_gram;  // Add the kgram to final string

  for (unsigned int a = 0; a < (L - (unsigned)order); a++) {
  return_char = k_Rand(final_string.substr(a, order));

    // Add the return_char to final_string
```

```
    final_string.push_back(return_char);

    // Keep looping til it stops.
  }

  return final_string;
}


  // Overload the stream insertion operator
  std::ostream& operator<< (std::ostream &out, RandWriter &randwri) {
  out << "\nOrder: " << randwri.order << "\n";
  out << "Alphabet: "<< randwri.alphabet << "\n";

  out << "Kgrams map: \n\n";

  std::map<std::string, int>::iterator it;

  for (it = randwri.kgramsMap.begin(); it != randwri.kgramsMap.end(); it++)
{
    out << it->first << "\t" << it->second << "\n";
  }

  return out;
}
```

```
// Copyright 2021 Raul Olivares
#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include "RandWriter.hpp"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/included/unit_test.hpp>

BOOST_AUTO_TEST_CASE(test1) {
  BOOST_REQUIRE_NO_THROW(RandWriter("ppplluuuttttttooo", 0));

  RandWriter test1("ppplluuuttttttooo", 0);

  BOOST_REQUIRE_THROW(test1.freq("g"), std::runtime_error);
  BOOST_REQUIRE_THROW(test1.freq("a"), std::runtime_error);
  BOOST_REQUIRE(test1.order_k() == 0);
  BOOST_REQUIRE(test1.freq("") == 16);  // length of input in constructor
  BOOST_REQUIRE(test1.freq("", 'p') == 3);
  BOOST_REQUIRE(test1.freq("", 'l') == 2);
  BOOST_REQUIRE(test1.freq("", 'u') == 3);
  BOOST_REQUIRE(test1.freq("", 't') == 5);
  BOOST_REQUIRE(test1.freq("", 'o') == 3);
  BOOST_REQUIRE(test1.freq("", 'g') == 0);
}

BOOST_AUTO_TEST_CASE(test2) {
  BOOST_REQUIRE_NO_THROW(RandWriter("gagggagaggcgagaaa", 5));

  RandWriter test2("gagggagaggcgagaaa", 5);

  BOOST_REQUIRE(test2.order_k() == 5);
  BOOST_REQUIRE_THROW(test2.freq("pl"), std::runtime_error);
  BOOST_REQUIRE_THROW(test2.freq("plu"), std::runtime_error);
  BOOST_REQUIRE_THROW(test2.freq("plut"), std::runtime_error);

  BOOST_REQUIRE(test2.freq("gaggg") == 1);
  BOOST_REQUIRE(test2.freq("ggaga") == 1);
  BOOST_REQUIRE(test2.freq("agagg") == 2);

  BOOST_REQUIRE(test2.freq("gaggg", 'a') == 1);
  BOOST_REQUIRE(test2.freq("ggaga", 'g') == 1);
  BOOST_REQUIRE(test2.freq("agagg", 'g') == 1);

  BOOST_REQUIRE_NO_THROW(test2.k_Rand("gaggg"));
  BOOST_REQUIRE_NO_THROW(test2.k_Rand("ggaga"));
  BOOST_REQUIRE_NO_THROW(test2.k_Rand("agagg"));

  BOOST_REQUIRE_THROW(test2.k_Rand("gggagg"), std::runtime_error);

  BOOST_REQUIRE_THROW(test2.k_Rand("gag"), std::runtime_error);
}

BOOST_AUTO_TEST_CASE(test3) {
  BOOST_REQUIRE_NO_THROW(RandWriter("plupluuutttooppp", 3));

  RandWriter test3("plupluuutttooppp", 3);

  BOOST_REQUIRE(test3.order_k() == 3);
```

```

  BOOST_REQUIRE_THROW(test3.freq("j"), std::runtime_error);
  BOOST_REQUIRE_THROW(test3.freq("n"), std::runtime_error);
  BOOST_REQUIRE_THROW(test3.freq("g", 'g'), std::runtime_error);
  BOOST_REQUIRE_THROW(test3.freq("gc", 'g'), std::runtime_error);

  BOOST_REQUIRE_NO_THROW(test3.freq("ooo"));

  BOOST_REQUIRE(test3.freq("plu") == 2);
  BOOST_REQUIRE(test3.freq("plu", 'p') == 1);
  BOOST_REQUIRE(test3.freq("uuu", 't') == 1);
  BOOST_REQUIRE(test3.freq("ppl", 't') == 0);
}
```

# PS7a: Kronos Time Clock

## The Assignment

This assignment asked us to use regular expressions from the Regex library to parse a log file from a Kronos InTouch time clock. To parse the file, I used Boost's regular expression library (regex), along with Boost's date_time libraries. I was required to create an output file that verifies whether a time clock successfully booted up or failed to boot. I was able to accomplish this and send the output into an rpt file.

## Key Concepts

The key concept in this assignment was the use of regular expressions. I used Boost's regex library to find matches in a document using a string containing regular expressions that I created. Another important concept was to use date and time libraries to deal with dates and regexs to scroll through the document. Boost's regex library was used by using boost::regex_match or boost::regex_search to find a match against the regular expressions previously created.

The following code was what I used to match start boots.

```
 // Regex used to search for matching lines in file for start of boot:
 // 2014-02-01 14:02:32: (log.c.166) server started
 // Success if we find:
 // "2014-01-26 09:58:04.362:INFO:oejs.AbstractConnector:Started
 // SelectChannelConnector@0.0.0.0:9080
 std::string startString = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
 startString += "([0-9]{2}):([0-9]{2}):([0-9]{2}): \\(log.c.166\\) ";
 startString += "server started";
 std::string endString = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
 endString += "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}):INFO:oejs.";
 endString += "AbstractConnector:Started
SelectChannelConnector@0.0.0.0:9080";
```

## What I Learned

I learned about both the Boost regex library and the Boost date_time libraries. I also learned about how to use various regular expressions that I had no idea about in the past. Regular seem like a very good idea to use to search for specific instances of certain things in any type of document that might be thousands and thousands of lines long. The date_time library seems efficient to use whenever a time or date is extremely important to the program.

## Created rpt file from program showing successful and attempted Boots:

```
Device Boot Report

Intouch log file: device1_intouch.log
Lines scanned: 443839

Device boot count: initiated = 6, completed: 6


=== Device boot ===
435369(device1_intouch.log): 2014-03-25 19:11:59 Boot Start
435759(device1_intouch.log): 2014-03-25 19:15:02 Boot Completed
      Boot Time: 183000ms

=== Device boot ===
436500(device1_intouch.log): 2014-03-25 19:29:59 Boot Start
436859(device1_intouch.log): 2014-03-25 19:32:44 Boot Completed
      Boot Time: 165000ms

=== Device boot ===
440719(device1_intouch.log): 2014-03-25 22:01:46 Boot Start
440791(device1_intouch.log): 2014-03-25 22:04:27 Boot Completed
      Boot Time: 161000ms

=== Device boot ===
440866(device1_intouch.log): 2014-03-26 12:47:42 Boot Start
441216(device1_intouch.log): 2014-03-26 12:50:29 Boot Completed
      Boot Time: 167000ms

=== Device boot ===
442094(device1_intouch.log): 2014-03-26 20:41:34 Boot Start
442432(device1_intouch.log): 2014-03-26 20:44:13 Boot Completed
      Boot Time: 159000ms

=== Device boot ===
443073(device1_intouch.log): 2014-03-27 14:09:01 Boot Start
443411(device1_intouch.log): 2014-03-27 14:11:42 Boot Completed
      Boot Time: 161000ms
```

```
# makefile for ps7
# Flags to save on typing all this out
CC = g++
CFLAGS = -g -Wall -Werror -std=c++17 -pedantic
SFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
Boost = -lboost_regex -lboost_date_time

# Make ps7
all:    ps7

# PS7 executable
ps7:    IntouchDevice.o
        $(CC) IntouchDevice.o -o ps7 $(Boost)

# Object files
IntouchDevice.o: IntouchDevice.cpp
        $(CC) -c IntouchDevice.cpp $(CFLAGS)

# Cleanup object files
clean:
        rm *.o
        rm ps7
```

```
// Copyright 2021 Raul Olivares
#include <iostream>
#include <fstream>
#include <string>
#include <boost/regex.hpp>
#include "boost/date_time/gregorian/gregorian.hpp"
#include "boost/date_time/posix_time/posix_time.hpp"

using boost::gregorian::date;
using boost::gregorian::from_simple_string;
using boost::gregorian::date_period;        // Need these for date
using boost::gregorian::date_duration;
using boost::posix_time::ptime;                // Need these for time
using boost::posix_time::time_duration;

int main(int argc, const char* argv[]) {
  if (argc != 2) {  // Shows needed input
    std::cout << "./ps7 device1_intouch.log\n";
    return 0;
  }

  int lineCounter = 1;  // Counters
  int bootSuccess = 0;
  int bootTotal = 0;

  std::string beginDate = "";  // Begin time date string
  std::string endDate = "";  // End time date string
  std::string completeDate;   // Date and time variables

  std::string fileName(argv[1]);  // File name from input
  std::string outputName = fileName + ".rpt";  // create needed output file
  std::string report = "";
  std::string boots = "";

  int hours = 0;
  int minutes = 0;
  int seconds = 0;

  ptime begin;  // timepoint manipulation variables
  ptime end;

  date failureDate;  // date variables to use gregorian date functions later
  date successDate;

  time_duration timeDifference;

  // Regex used to search for matching lines in file for start of boot:
  // 2014-02-01 14:02:32: (log.c.166) server started
  // Success if we find:
  // "2014-01-26 09:58:04.362:INFO:oejs.AbstractConnector:Started
  // SelectChannelConnector@0.0.0.0:9080
  std::string startString = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
  startString += "([0-9]{2}):([0-9]{2}):([0-9]{2}): \\(log.c.166\\) ";
  startString += "server started";
  std::string endString = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
  endString += "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}):INFO:oejs.";
  endString += "AbstractConnector:Started SelectChannelConnector@0.0.0.0:908
0";

  // Make two regexes
  boost::regex startRegex(startString, boost::regex::perl);
```

```
boost::regex endRegex(endString);

// Use this for getting parts of the matched string.
boost::smatch matches;

std::string line;
std::ifstream file(fileName.c_str());  // read file

// Need to keep track of when we've found a start.
bool foundBoot = false;

if (file.is_open()) {
  while (getline(file, line)) {
    // We've got the current string here and can do stuff with it.

    beginDate.clear();  // clear begin/end strings
    endDate.clear();

    // Let's try and see if we found a start boot.
    if (boost::regex_search(line, matches, startRegex)) {
      // Get the start time information
      beginDate = matches[1] + "-" + matches[2] + "-" + matches[3];
      beginDate += " " + matches[4] + ":" + matches[5] + ":" + matches[6];

      completeDate = matches[1] + "-" + matches[2] + "-" + matches[3];
      failureDate = date(from_simple_string(completeDate));

      hours = std::stoi(matches[4]);
      minutes = std::stoi(matches[5]);
      seconds = std::stoi(matches[6]);

      begin = ptime(failureDate, time_duration(hours, minutes, seconds));

      // this is an incomplete boot whenever foundBoot is true
      if (foundBoot == true) {
        boots += "**** Incomplete boot **** \n\n";
      }

      // Now we want to add this to the output string as boot start.
      boots += "=== Device boot ===\n";
      boots += std::to_string(lineCounter) + "(" + fileName + "): ";
      boots += beginDate + " Boot Start\n";

      bootTotal++;

      // Match this as a "foundBoot"
      foundBoot = true;
    }

    // For successful boots
     if (boost::regex_match(line, matches, endRegex)) {
      // Get the end time, save it for later.
      endDate = matches[1] + "-" + matches[2] + "-" + matches[3];
      endDate += " " + matches[4] + ":" + matches[5] + ":" + matches[6];

      completeDate = matches[1] + "-" + matches[2] + "-" + matches[3];
      successDate = date(from_simple_string(completeDate));

      hours = std::stoi(matches[4]);
      minutes = std::stoi(matches[5]);
      seconds = std::stoi(matches[6]);
```

```

        end = ptime(successDate, time_duration(hours, minutes, seconds));

        // Add the end boot line and total time it took to get here.
        boots += std::to_string(lineCounter) + "(" + fileName + "): ";
        boots += endDate + " Boot Completed\n";

auto BOOTTIMEEQUATION = [&]() {return (end-begin);};
timeDifference = BOOTTIMEEQUATION();  // calculate boot time

        boots += "\tBoot Time: ";
     boots += std::to_string(timeDifference.total_milliseconds()) + "ms \n\n";
       // boots += "Services\n\n";
       // boots += "\tNeed Regex\n\n";   was going to use for services section
       // boots += "\t\tStart:\n\n";
       // boots += "\t\tCompleted:\n\n";
       // boots += "\t\tElapsed Time:\n\n";

        bootSuccess++;
        foundBoot = false;  // prepare next loop
       }

      lineCounter++;
    }
    file.close();
  }

  // Add lines scanned to the report.
  report += "Device Boot Report";
  report += "\n\nIntouch log file: " + fileName + "\n";
  report += "Lines scanned: " + std::to_string(lineCounter) + "\n\n";

  // show amount of boots attempted and amount completed
  report += "Device boot count: initiated = " + std::to_string(bootTotal);
  report += ", completed: " + std::to_string(bootSuccess) + "\n\n\n";

  report += boots;  // adds boots section to report

  std::ofstream out(outputName.c_str());  // creates rpt file
  out << report;
  out.close();

  return 0;
}
```