

Your Name: Doulos Htet
COMP IV Sec 203: Project Portfolio
Spring 2022

Contents:

PS0 Hello World with SFML

PS1 Linear Feedback Shift Register and Image Encoding

PS2 N-Body Simulation

PS3 Recursive Graphics

PS4 Synthesizing a Plucked String Sound

PS5 DNA Sequence Alignment

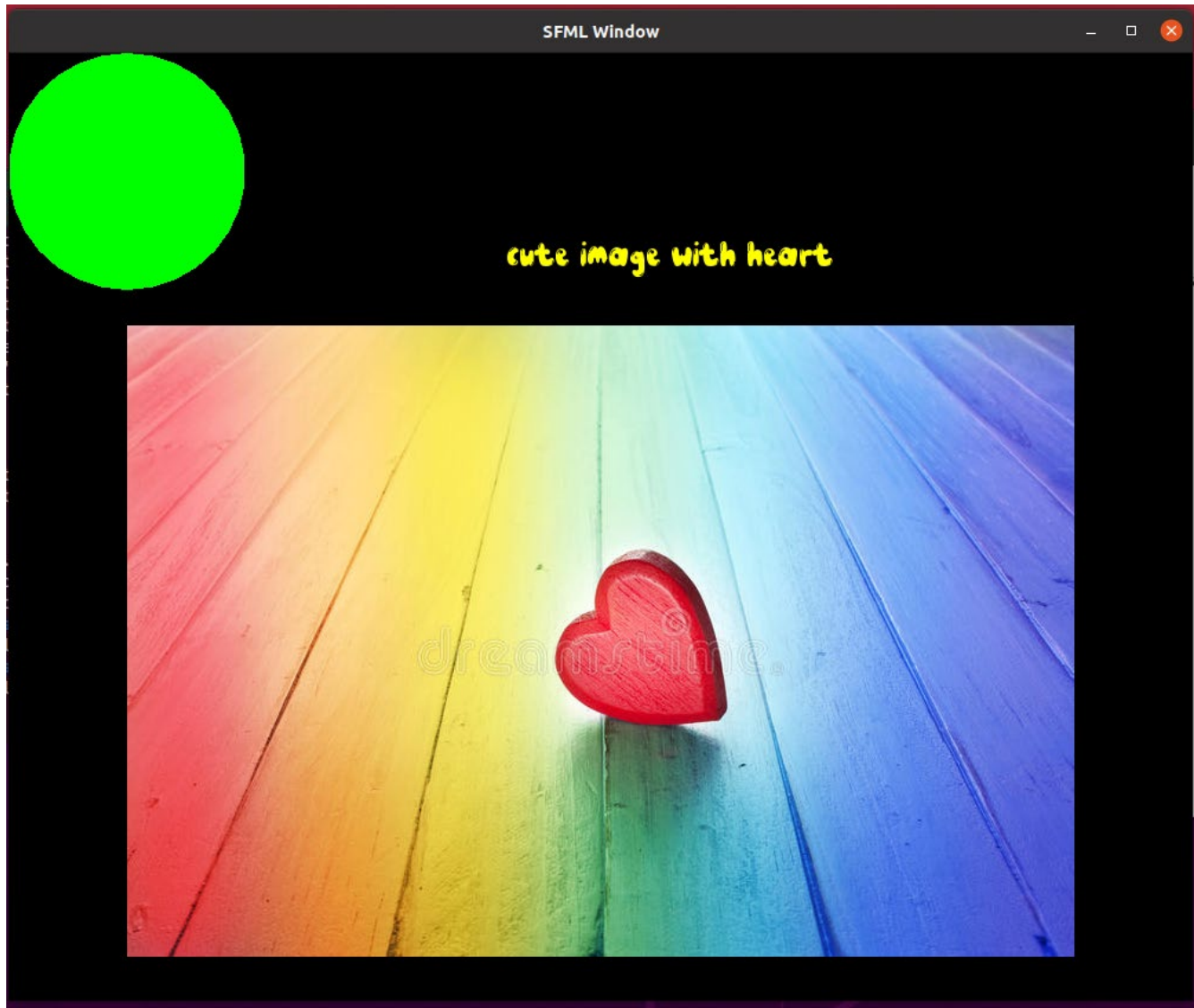
PS6 Random Writer

PS7 Kronos Time Clock

Time to complete: 5 hours

PS0: Hello World with SFML

This is the first assignment that introduces to SFML libraries. The assignment sets up the SFML environment that students will be using for the rest of the semester. The assignment is named Hello World with SFML and the task is to create an SFML window and load an image file in it with the extra elements being the use of keyboard arrows to move the image file. I use the SFML objects: Sprite and Texture to draw the image of the green circle as well as load the external image file. The circle and image files are loaded in a window with given fixed size as well as the text “cute image with heart” with JustBubble font. The keyboard SFML isKeyPressed function is used to move the image with the corresponding arrow keys. From this assignment, I learned how to use functions from SFML libraries and implement them.



Files:

Main.cpp

```

2  * Name: Doulos Htet
3  *
4  * Course Name: Computing IV
5  *
6  * Assignment #: PS0
7  *
8  * Instructor: Dr. Yelena Rykalova
9  *
10 * Due Date: 1/24/2022
11 *
12 * Problem: Learning to compile the cpp program was head-scratching. I was trying with visual studio first
13 * and later found out the easy way to compile in a specific directory with terminal
14 *
15 * Bugs: code works fine
16 *
17 */
18
19 #include <SFML/Graphics.hpp>
20
21 //main start
22 int main()
23 {
24     // display window
25     sf::RenderWindow window(sf::VideoMode(1000, 800), "SFML Window");
26     sf::CircleShape shape(100.f);
27     shape.setFillColor(sf::Color::Green);
28
29     // display sprite
30     sf::Texture texture;
31     if (!texture.loadFromFile("sprite.jpg"))
32         return EXIT_FAILURE;
33     sf::Sprite sprite(texture);
34     sprite.setPosition(sf::Vector2f(100, 230));
35
36     // display text
37     sf::Font font;
38     if (!font.loadFromFile("JustBubble.ttf"))
39         return EXIT_FAILURE;
40     sf::Text text("cute image with heart", font, 30);
41     text.setFillColor(sf::Color::Yellow);
42     text.setPosition(420, 150);
43
44     // Loop to display windows
45     while (window.isOpen())
46     {
47         sf::Event event;
48         while (window.pollEvent(event))
49         {
50             if (event.type == sf::Event::Closed)
51                 window.close();
52         }
53
54         // codes to move the sprite with arrow keys
55         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
56         {
57             sprite.move(sf::Vector2f(0, -0.5));
58         }
59
60         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
61         {
62             sprite.move(sf::Vector2f(0, 0.5));
63         }
64
65         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
66         {
67             sprite.move(sf::Vector2f(0.5, 0));
68         }
69
70         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
71         {
72             sprite.move(sf::Vector2f(-0.5, 0));
73         }
74
75         // clear, draw, and display the window
76         window.clear();
77         window.draw(shape);
78         window.draw(sprite);
79         window.draw(text);
80         window.display();
81
82     }
83
84     return 0;
85 }

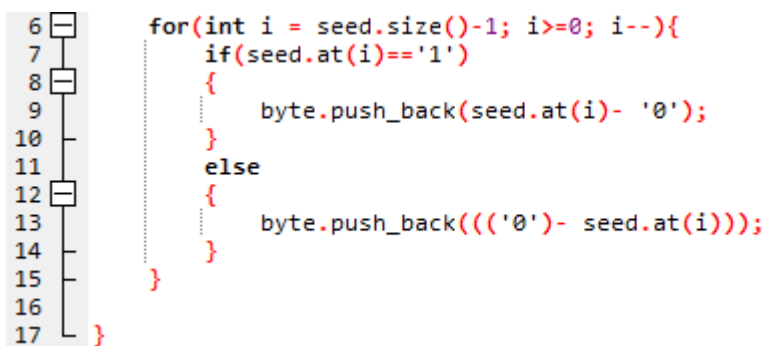
```

PS1: Linear Feedback Shift Register and Image Encoding

Part A: Linear Feedback Shift Register

The assignment PS1 has two parts- A and B. The first part PS1A is about using the XOR operator on the specific bits and shifting them to the left one by one. The assignment instructs to take the left-most bit from each binary seed (16 bits) and XOR it with the different bits at nth position from the seed and adding the result at the right end of the seed. Then, the left-most bit is removed as the result is added at the right end, thus giving as the new seed. This process is performed by function step() in the program. The function step() uses the left-most bit and XOR it with the 15th bit, then 13th bit, 12th bit and lastly, 10th bit from the seed. When the LSFR object is created with given seed, each bit is stored in a vector named byte. The function generate(int k) takes in specific number of times to run the step function. For example, if the generate takes in 5, the step() function is run five times meaning that the bit shift occurred five times to the left. After the run is done, the generate function returns the binary seed as decimal form. The results are printed by Ostream operator. The bits are stored in the vector reversely so when the results are printed the seed given aligns with the nth index of the vector.

This is for storing the bits.



The diagram shows a vertical shift register with 16 bits, indexed from 6 at the top to 17 at the bottom. Bits 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, and 17 are shown. Bits 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, and 17 are shown. Bits 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, and 17 are shown. The code snippet is as follows:

```
for(int i = seed.size()-1; i>=0; i--){
    if(seed.at(i)=='1')
    {
        byte.push_back(seed.at(i)- '0');
    }
    else
    {
        byte.push_back((( '0')- seed.at(i)));
    }
}
```

This is for XOR with the 15th, 13th, 12th, 10th bits.

```
bit = byte.at(byte.size()-1) ^ (byte.at(13)) ^ (byte.at(12)) ^ (byte.at(10));
```

This is the first time that I learned Boost test framework and I tested to see if the given seed running step() function has the same result. Also, I tested to see if generating a specific number of times gives the resulted seed in binary form. I did the math by myself and see if the boost test fails or succeeds.

```

1  #include <iostream>
2  #include <string>
3  #include <sstream>
4  #include "FibLFSR.h"
5
6  #define BOOST_TEST_DYN_LINK
7  #define BOOST_TEST_MODULE Main
8  #include <boost/test/unit_test.hpp>
9
10 BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
11     FibLFSR l("1011011000110110");
12     BOOST_REQUIRE(l.step() == 0);
13     BOOST_REQUIRE(l.step() == 0);
14     BOOST_REQUIRE(l.step() == 0);
15     BOOST_REQUIRE(l.step() == 1);
16     BOOST_REQUIRE(l.step() == 1);
17     BOOST_REQUIRE(l.step() == 0);
18     BOOST_REQUIRE(l.step() == 0);
19     BOOST_REQUIRE(l.step() == 0);
20     BOOST_REQUIRE(l.step() == 1);
21
22     FibLFSR l2("1011011000110110");
23     BOOST_REQUIRE(l2.generate(9) == 51);
24 }
25 //test1 with returning XOR bit
26 BOOST_AUTO_TEST_CASE(test1) {
27     FibLFSR test1("1011011000110110");
28     BOOST_REQUIRE(test1.step() == 0);
29     BOOST_REQUIRE(test1.step() == 0);
30     BOOST_REQUIRE(test1.step() == 0);
31     BOOST_REQUIRE(test1.step() == 1);
32     BOOST_REQUIRE(test1.step() == 1);
33     BOOST_REQUIRE(test1.step() == 0);
34     BOOST_REQUIRE(test1.step() == 0);
35     BOOST_REQUIRE(test1.step() == 0);
36     BOOST_REQUIRE(test1.step() == 1);
37 }
38 //test2 with testing if the last 5 bits equal to the decimal representation
39 BOOST_AUTO_TEST_CASE(test2) {
40     FibLFSR test2("1011011000110110");
41     BOOST_REQUIRE(test2.generate(5) == 3);
42
43
44
45 }

```

Part B: Image Encoding

For part B of the assignment, the function PhotoMagic.cpp was added to extend part A. The instructions of this part is to encrypt and decrypt a given image and seed. The functions in SFML libraries- Image, Vector2u, Texture, Sprite are used in PhotoMagic.cpp. Image and Vector2u store the information of image input and Texture and Sprite load the images in the window. Based on the input image, the pixels red, green and blue are extracted and FibLFSR object is called to form a new 8 bit-integer. The integer is stored in each color of the image, thus resulting in different pixel in the image. The for-loop function loops through each pixel of the image and transforming the image as a whole into a new one. Two separate windows to show the encoded and decoded images. The result image is also based on the input seed. The same seed is used to decrypt the resulted image into original image.

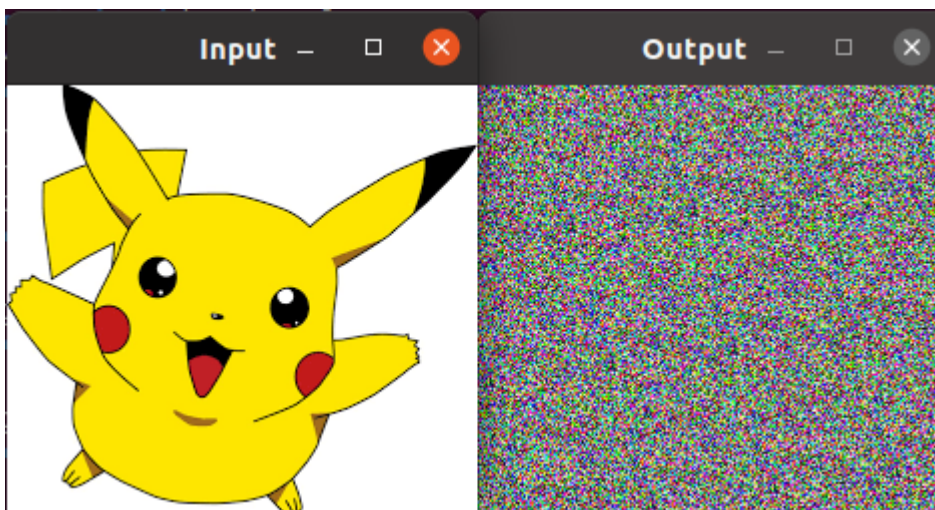
The for-loop function to change each rgb pixels.

```

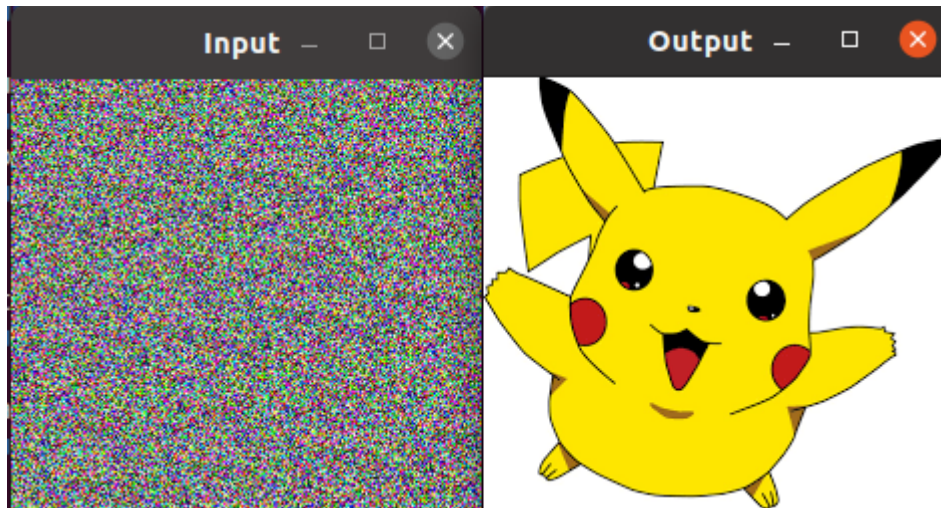
23   for(int x= 0;x<(int)image.getSize().x;x++) {
24       for (int y = 0; y< (int)image.getSize().y; y++) {
25           p = pic->getPixel(x,y);
26           p.r = 255 - p.r;
27           p.g = 255 - p.g;
28           p.b = 255 - p.b;
29
30           p.r = p.r - byte->generate(10);
31           p.g = p.g - byte->generate(10);
32           p.b = p.b - byte->generate(10);
33
34           pic->setPixel(x,y,p);
35       }
36   }

```

Encoded image



Decoded image



Files:

Makefile

FibLFSR.h

FibLFSR.cpp

PhotoMagic.cpp

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

```

1  #ifndef FIBLFSR_H
2  #define FIBLFSR_H
3
4  #include <iostream>
5  #include <vector>
6  #include <string>
7  #include <SFML/System.hpp>
8  #include <SFML/Window.hpp>
9  #include <SFML/Graphics.hpp>
10 #include <sstream>
11
12 class FibLFSR {
13 public:
14
15     FibLFSR(std::string seed);           //constructor to create LFSR with the given initial seed
16     int step();                         // simulate one step and return the new bit as 0 or 1
17     int generate(int k);                // simulate k steps and return k-bit integer
18
19     friend std::ostream& operator<<(std::ostream &outStream, const FibLFSR &b);
20     std::vector<int> byte;
21
22 private:
23     std::string seed;
24     int tap,bit;
25
26 };
27
28 #endif //FIBLFSR_H

```



```

1  #include "FibLFSR.h"
2  #include <iostream>
3  #include <sstream>
4  #include <string>
5  using namespace std;
6
7  FibLFSR::FibLFSR(string seed) {           //take in seed
8      for(int i = seed.size()-1; i>=0; i--){
9          if(seed.at(i)=='1')
10             {
11                 byte.push_back(seed.at(i)- '0');
12             }
13             else
14             {
15                 byte.push_back((( '0' )- seed.at(i)));
16             }
17         }
18     }
19
20     int FibLFSR::step() {                  //XOR 13,12,10
21
22         bit = byte.at(byte.size()-1) ^ (byte.at(13)) ^ (byte.at(12)) ^ (byte.at(10));
23         byte.pop_back();
24         byte.insert(byte.begin(), bit);
25         return bit;
26     }
27
28
29     int FibLFSR::generate(int k) {         //binary to integer
30
31         int number = 0;
32         for(int i=0; i<k; i++){
33             number *=2;
34             number += step();
35         }
36         return number;
37     }
38
39     ostream& operator<<(ostream &outStream, const FibLFSR &b ){ //output operator
40         for(int i = b.byte.size()-1; i>= 0; --i)
41         {
42             outStream << (b.byte[i]);
43         }
44         return outStream;
45     }

```

```

1 //header files
2 #include <SFML/System.hpp>
3 #include <SFML/Window.hpp>
4 #include <SFML/Graphics.hpp>
5 #include "FibLFSR.h"
6 #include <sstream>
7
8 //transforms image using FibLFSR
9 void transform(sf::Image *pic, FibLFSR *byte){
10     sf::Image image = *pic;
11     sf::Vector2u size = pic->getSize();
12     sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Input");
13
14     sf::Texture texture;
15     texture.loadFromImage(image);
16
17     sf::Sprite sprite;
18     sprite.setTexture(texture);
19
20     sf::Color p;
21
22     // create photographic negative image of upper-left image px square
23     for(int x= 0; x<(int)image.getSize().x; x++) {
24         for (int y = 0; y< (int)image.getSize().y; y++) {
25             p = pic->getPixel(x,y);
26             p.r = 255 - p.r;
27             p.g = 255 - p.g;
28             p.b = 255 - p.b;
29
30             p.r = p.r - byte->generate(10);
31             p.g = p.g - byte->generate(10);
32             p.b = p.b - byte->generate(10);
33
34             pic->setPixel(x,y,p);
35         }
36     }
37     //output image rendering
38     sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "Output");
39
40     sf::Texture texture2;
41     texture2.loadFromImage(*pic);
42
43     sf::Sprite sprite2;
44     sprite2.setTexture(texture2);
45

```

```

46 while (window1.isOpen() && window2.isOpen())
47 {
48     sf::Event event;
49     while (window1.pollEvent(event))
50     {
51         if (event.type == sf::Event::Closed)
52             window1.close();
53     }
54     while(window2.pollEvent(event)){
55         if(event.type == sf::Event::Closed)
56             window2.close();
57     }
58
59     window1.clear();
60     window1.draw(sprite);
61     window1.display();
62     window2.clear();
63     window2.draw(sprite2);
64     window2.display();
65 }
66
67 }
68
69 //main function
70 int main(int argc, char* argv[])
71 {
72     FibLFSR l = FibLFSR(argv[3]);
73     sf::Image image;
74
75     if(!image.loadFromFile(argv[1]))
76     {
77         return -1;
78     }
79     else
80     {
81         transform(&image, &l);
82     }
83
84     if(!image.saveToFile(argv[2]))
85     {
86         std::cout << "unable to save" << std::endl;
87     }
88
89     return 0;
90 }

```

PS2: N-Body Simulation

Part A:

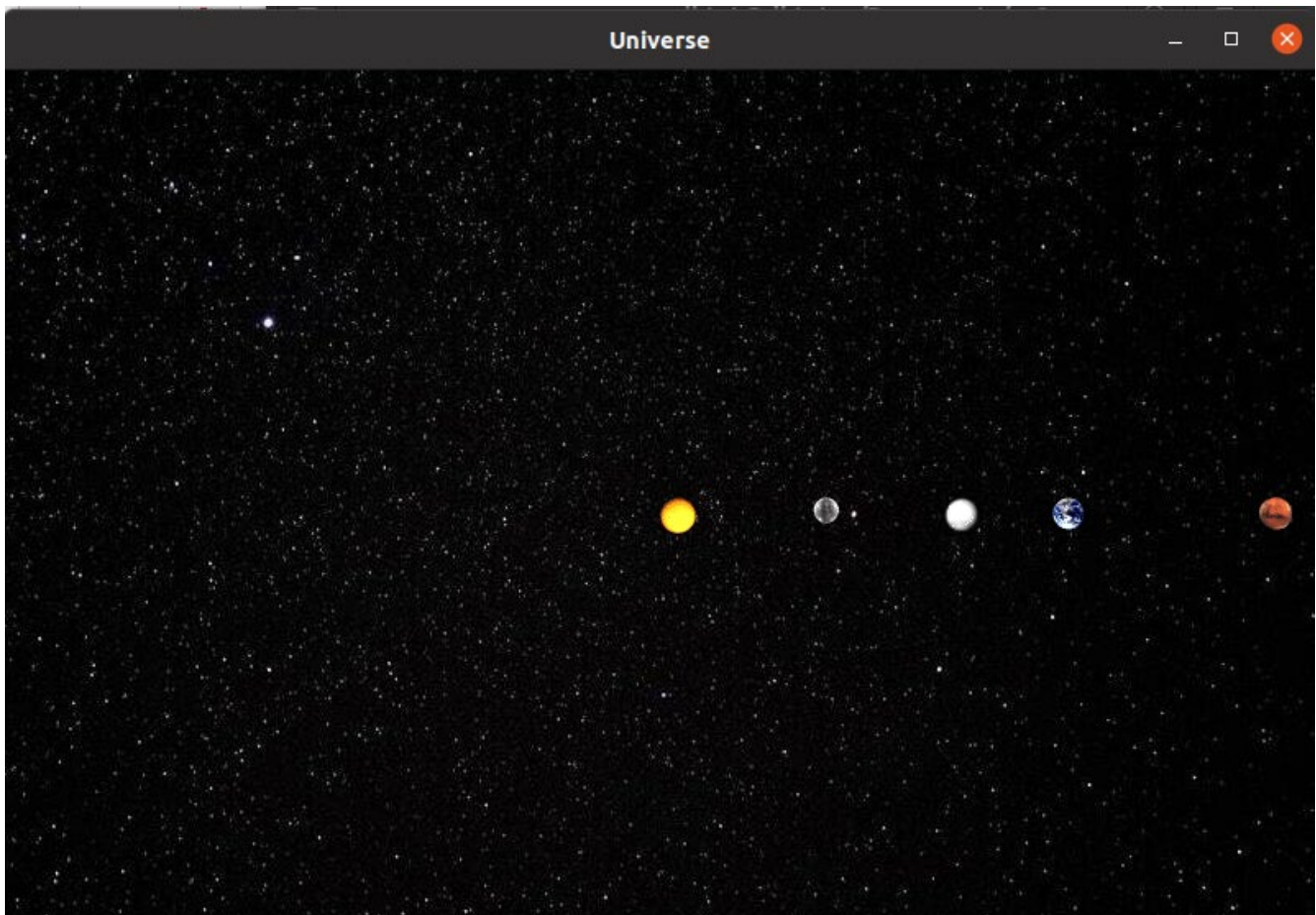
The assignment PS2 has two parts A and B. The goal of this assignment is to simulate a universe with celestial bodies in SFML window. The instruction includes parsing given files and placing the planets(including sun) at specific positions. The user is provided with a file that contains number of particles, the radius of the planet and more information on the planet. The information are xpos, ypos, xvel, yvel, mass and filename. The values xpos and ypos determines the x and y coordinates of the planet to be placed in window. The values xvel and yvel are the velocities of the planet. The mass value is the mass of the planet and the filename is the name of the planet, e.g., Earth, Mercury. The images of the planets are loaded by Texture and Sprite. When the file with information is parsed, Universe object is created and readfile function is called to create CelestialBodies at given information. The CelestialBodies are store in shared vector pointer and drawUniverse function draws each planet in window. The background image is for the extra credit and is loaded with Texture. From this assignment, I learned about shared pointers and parsing input files.

Parsing files and creating CelestialBodies in a shared pointer.

```

3 void Universe::readFile(){
4     std::cin >> numberOfPlanets;
5     std::cin >> radius;
6
7     for(int i =0; i< numberOfPlanets; ++i){
8         std::shared_ptr<CelestialBody> planet(new CelestialBody(radius));
9         std::cin >> *planet;
10        SolarSystem.push_back(planet);
11
12    }
13
14 }
15

```



Part B:

In part B of the assignment, the information for mass, velocity is utilized to create moving planets in Universe. The CelestialBodies revolve around the Sun and each has different properties- mass, velocity, net force, acceleration, delta x and y. According to the information, the planets revolve around the Sun at specific speed. The background music is also opened from using Music from SFML library. From this assignment, I learned about unique pointers and the physics implementation in programming.

Files:

Makefile

CelestialBody.h

CelestialBody.cpp

Universe.h

Universe.cpp

main.cpp

```
1 CC = g++
2 CFLAGS = -c
3
4 run: main.o Universe.o CelestialBody.o
5     $(CC) main.o Universe.o CelestialBody.o -o NBody -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
6
7 main.o: main.cpp
8     $(CC) $(CFLAGS) main.cpp
9
10 Universe.o: Universe.cpp
11     $(CC) $(CFLAGS) Universe.cpp
12
13 CelestialBody.o: CelestialBody.cpp
14     $(CC) $(CFLAGS) CelestialBody.cpp
15
16 clean:
17     rm -rf *o NBody
```

```

1  #ifndef CELESTIALBODY_H
2  #define CELESTIALBODY_H
3
4  #include <cmath>
5  #include <memory>
6  #include <string>
7  #include <stdio.h>
8  #include <fstream>
9  #include <iostream>
10 #include <SFML/System.hpp>
11 #include <SFML/Window.hpp>
12 #include <SFML/Graphics.hpp>
13
14 #define G 6.67e-11
15
16 class CelestialBody: public sf::Drawable
17 {
18     public:
19         CelestialBody(double);
20         CelestialBody(double, double, double, double, double, std::string);
21
22         void windowSize(double, double);
23         void draw(sf::RenderTarget &target, sf::RenderStates states) const;
24         friend std::ostream& operator <<(std::ostream&, CelestialBody&);
25         friend std::istream& operator >>(std::istream&, CelestialBody&);
26
27         void setMass(double);
28         void setxPos(double);
29         void setyPos(double);
30         void setxVel(double);
31         void setyVel(double);
32         void setDeltaX(double);
33         void setDeltaY(double);
34         void setxForce(double);
35         void setyForce(double);
36         void setxAcceleration();
37         void setyAcceleration();
38         void setNetForce(double, double);
39
40         double getMass();
41         double getxPos();
42         double getyPos();
43         double getxVel();
44         double getyVel();
45         double getDeltaX();
46         double getDeltaY();
47         double getxForce();
48         double getyForce();
49         double getxAcceleration();
50         double getyAcceleration();
51         double getNetForce();
52
53     private:
54
55         sf::Sprite sprite;
56         sf::Texture texture;
57         sf::Image image;
58         std::string planetName;
59         double mass, radius, xPos, xVel, yPos, yVel, distance, winX, winY;
60         double deltaTime, xForce, yForce, xAcc, yAcc, deltaX, deltaY, NetForce;
61
62 };
63 #endif
64 //CELESTIALBODY_H

```



```

1  #include "CelestialBody.h"
2  using namespace sf;
3
4  //consturtor with one parameter
5  CelestialBody::CelestialBody(double radius){
6      this->radius = radius;
7  }
8
9  //constructor with parameters
10 CelestialBody::CelestialBody(double xPos, double yPos, double xVel, double yVel, double, std::string planetName)
11 {
12     this->xPos = xPos;
13     this->yPos = yPos;
14     this->xVel = xVel;
15     this->yVel = yVel;
16     this->planetName = planetName;
17 }
18
19 //window size
20 void CelestialBody::windowSize(double x, double y){
21     winX = x;
22     winY = y;
23 }
24
25 //overriden operator for inputting files
26 std::istream& operator >> (std::istream &input, CelestialBody &cBody){
27     input >> cBody.xPos >> cBody.yPos >> cBody.xVel >> cBody.yVel >> cBody.mass >> cBody.planetName;
28
29     if(!cBody.image.loadFromFile(cBody.planetName))
30     {
31         std::cout << "Error" << std::endl;
32     }
33
34     cBody.texture.loadFromImage(cBody.image);
35     cBody.sprite.setTexture(cBody.texture);
36     return input;
37 }
38
39 //draw x and y position on window
40 void CelestialBody::draw(sf::RenderTarget &target, sf::RenderStates states) const
41 {
42     double scaled = (radius*2)/winX;
43     double x = xPos/scaled;
44     double y = yPos/scaled;
45
46     sf::Sprite sprite2 = sprite;
47     sprite2.setPosition(x+(winX/2),y+(winY/2));
48     target.draw(sprite2);
49 }
50
51 //overriden operator for outputting files
52 std::ostream& operator <<(std::ostream &os, CelestialBody &cBody)
53 {
54     os << "PlanetName:" << cBody.planetName << std::endl;
55     os << "Pos x:" << cBody.xPos << std::endl;
56     os << "Pos y:" << cBody.yPos << std::endl;
57     os << "Vel x:" << cBody.xVel << std::endl;
58     os << "Vel y:" << cBody.yVel << std::endl;
59     os << "Mass" << cBody.mass << std::endl;
60     os << "Radius" << cBody.radius << std::endl;
61     os << "DeltaTime" << cBody.deltaTime << std::endl;
62     os << "XForce" << cBody.xForce << std::endl;
63     os << "YForce" << cBody.yForce << std::endl;
64     os << "XAcceleration" << cBody.xAcc << std::endl;
65     os << "YAcceleration" << cBody.yAcc << std::endl;
66     return os;
67 }
68
69 void CelestialBody::setxAcceleration(){
70     xAcc = xForce / mass;
71 }
72
73 void CelestialBody::setyAcceleration(){
74     yAcc = yForce / mass;
75 }
76
77 void CelestialBody::setxForce(double xForce){
78     this->xForce = xForce;
79 }

```

```

80
81 void CelestialBody::setyForce(double yForce){
82     this->yForce = yForce;
83 }
84
85 void CelestialBody::setMass(double mass) {
86     this->mass = mass;
87 }
88
89 void CelestialBody::setxPos(double xPos){
90     this->xPos = xPos;
91 }
92
93 void CelestialBody::setyPos(double yPos){
94     this->yPos = yPos;
95 }
96
97 void CelestialBody::setxVel(double deltaTime){
98     xVel = xVel + (deltaTime * xAcc);
99 }
100
101 void CelestialBody::setyVel(double deltaTime){
102     yVel = yVel - (deltaTime * yAcc);
103 }
104
105 void CelestialBody::setNetForce(double prevMass, double r){
106     NetForce = ((G*mass*prevMass)/r);
107 }
108
109 double CelestialBody::getxAcceleration(){
110     return xAcc;
111 }
112
113 double CelestialBody::getyAcceleration(){
114     return yAcc;
115 }
116
117 double CelestialBody::getxForce(){
118     return xForce;
119 }
120
121 double CelestialBody::getyForce(){
122     return yForce;
123 }
124
125 double CelestialBody::getMass(){
126     return mass;
127 }
128
129 double CelestialBody::getxPos(){
130     return xPos;
131 }
132
133 double CelestialBody::getyPos(){
134     return yPos;
135 }
136
137 double CelestialBody::getxVel(){
138     return xVel;
139 }
140
141 double CelestialBody::getyVel(){
142     return yVel;
143 }
144
145 double CelestialBody::getNetForce(){
146     return NetForce;
147 }

```

```

1  #ifndef UNIVERSE_H
2  #define UNIVERSE_H
3
4  #include <cmath>
5  #include <iostream>
6  #include <fstream>
7  #include <vector>
8  #include <memory>
9  #include <SFML/Audio.hpp>
10 #include "CelestialBody.h"
11
12 class Universe
13 {
14     public:
15         void readFile();
16         void drawUniverse();
17         void step(double, double);
18
19     private:
20         sf::Sprite sprite;
21         sf::Texture texture;
22         int numberOfPlanets;
23
24         double newYPos, newXPos;
25         double radius, xForce, yForce, deltaX, deltaY, r;
26
27         std::vector<std::shared_ptr<CelestialBody> > SolarSystem;
28 };
29
30 #endif
31 //UNIVERSE_H

```

```

1  #include "Universe.h"
2
3  void Universe::readFile(){
4      std::cin >> numberOfPlanets;
5      std::cin >> radius;
6
7      for(int i =0; i< numberOfPlanets; ++i){
8          std::shared_ptr<CelestialBody> planet(new CelestialBody(radius));
9          std::cin >> *planet;
10         SolarSystem.push_back(planet);
11     }
12 }
13
14 }
15
16 void Universe::drawUniverse(){
17     if(!texture.loadFromFile("background.jpg")){
18         std::cout << "no background file" << std::endl;
19     }
20
21     sf::RenderWindow window(sf::VideoMode(texture.getSize().x, texture.getSize().y), "Universe");
22     sprite.setTexture(texture);
23
24     while(window.isOpen()){
25         sf::Event event;
26
27         while (window.pollEvent(event)){
28             if (event.type == sf::Event::Closed){
29                 window.close();
30             }
31         }
32
33         window.draw(sprite);
34
35         for(int i=0;i<numberOfPlanets; ++i){
36             //(*SolarSystem.at(i)).windowSize(texture.getSize().x, texture.getSize().y);
37             window.draw(*SolarSystem.at(i));
38         }
39
40
41         window.display();
42     }
43 }
44

```

```

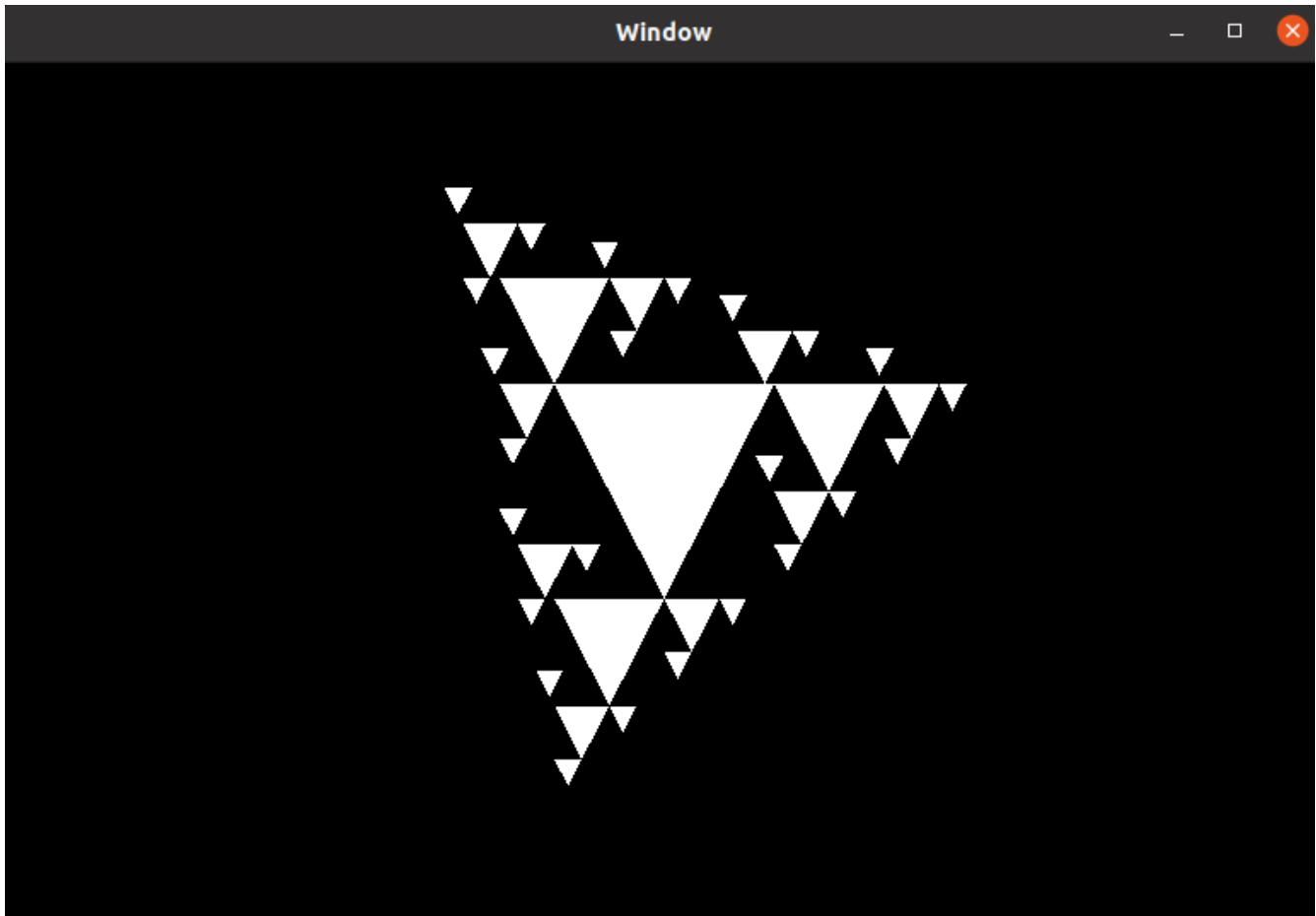
45 void Universe::step(double time, double deltaTime){
46     if(!texture.loadFromFile("background.jpg")){
47         std::cout << "no background file" << std::endl;
48     }
49
50     sf::RenderWindow window(sf::VideoMode(texture.getSize().x, texture.getSize().y), "Universe");
51     sprite.setTexture(texture);
52     sf::Music music;
53     if(!music.openFromFile("music.ogg"))
54     {
55         std::cout << "no music" << std::endl;
56     }
57     music.play();
58
59     for(int i = 0; i < time; i += deltaTime){
60         sf::Event event;
61
62         while (window.pollEvent(event)){
63             if (event.type == sf::Event::Closed){
64                 window.close();
65             }
66         }
67         window.draw(sprite);
68
69         for (int j = 0; j < numberOfPlanets; ++j){
70             for(int k = 0; k < numberOfPlanets; ++k){
71                 if(j != k){
72                     (*(SolarSystem).at(j)).windowSize(texture.getSize().x, texture.getSize().y);
73
74                     deltaX = (*(SolarSystem).at(k)).getxPos() - (*(SolarSystem).at(j)).getxPos();
75                     deltaY = (*(SolarSystem).at(k)).getyPos() - (*(SolarSystem).at(j)).getyPos();
76
77                     r = sqrt((deltaX * deltaX) + (deltaY * deltaY));
78
79                     (*(SolarSystem).at(j)).setNetForce((*(SolarSystem).at(k)).getMass(), r * r);
80
81                     xForce = (*(SolarSystem).at(j)).getNetForce() * (deltaX / r);
82                     yForce = (*(SolarSystem).at(j)).getNetForce() * (deltaY / r);
83
84                     (*(SolarSystem).at(j)).setxForce(xForce);
85                     (*(SolarSystem).at(j)).setyForce(yForce);
86
87                     (*(SolarSystem).at(j)).setxAcceleration();
88                     (*(SolarSystem).at(j)).setyAcceleration();
89
90                     (*(SolarSystem).at(j)).setxVel(deltaTime);
91                     (*(SolarSystem).at(j)).setyVel(deltaTime);
92                 }
93             }
94
95             newXPos = (*(SolarSystem).at(j)).getxPos() + (deltaTime * (*(SolarSystem).at(j)).getxVel());
96             newYPos = (*(SolarSystem).at(j)).getyPos() - (deltaTime * (*(SolarSystem).at(j)).getyVel());
97
98             (*(SolarSystem).at(j)).setxPos(newXPos);
99             (*(SolarSystem).at(j)).setyPos(newYPos);
100             window.draw(*(SolarSystem).at(j));
101         }
102         window.display();
103     }
104     //drawUniverse();
105 }
106

```

```
1 #include <iostream>
2 #include "Universe.h"
3 #include "CelestialBody.h"
4
5 int main(int argc, char** argv){
6     Universe universe;
7     universe.readFile();
8
9     float t = atof(argv[1]);
10    float dt = atof(argv[2]);
11
12    double time = (double) t;
13    double deltaTime = (double) dt;
14
15    universe.step(time, deltaTime);
16
17    //universe.drawUniverse();
18
19 }
```

PS3: Recursive Graphics

The assignment PS3 is about using recursive function to create the Sierpinski Triangle, which is a tree made up of recursive triangles using SFML libraries and functions. The tree is called TFractal and the program accepts the length of a side of the recursive triangle and the number of times to be repeated. The TFractal uses the information from the previous leaf to build the next leaf on the tree. The calculation is based on the user input of the length of a side of the main leaf (base triangle). Starting from the base triangle, the calculation is done based for the next leaf and it is recursively drawn until the number of times to be repeated is met.



The base triangle has its top, bottom, and right triangles. Those three triangles' sides are based on the base triangle's side divided by 2. From there, recursive functions are performed to form a full Sierpinski Triangle with the number of repeated times from the user input. From this assignment, I learned about the recursion to the next level and the use of the Pythagorean theorem.

Files:

Makefile

main.cpp

TFractal.h

TFractal.cpp

Triangle.h

Triangle.cpp

```
1 CC = g++
2 CFLAGS = -std=c++11 -Wall -Werror -pedantic
3 SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
4
5 all:: TFractal
6
7 TFractal: main.o TFractal.o Triangle.o
8     $(CC) main.o TFractal.o Triangle.o -o TFractal $(SFMLFLAGS)
9
10 TFractal.o: TFractal.cpp Triangle.cpp
11     $(CC) -c $(CFLAGS) TFractal.cpp Triangle.cpp
12
13 Triangle.o: Triangle.cpp
14     $(CC) -c $(CFLAGS) Triangle.cpp
15
16 main.o: main.cpp
17     $(CC) -c $(CFLAGS) main.cpp
18
19 clean:
20     rm *.o
21     rm TFractal
```

```

1  #include <iostream>
2  #include <stdexcept>
3  #include <sstream>
4  #include <SFML/Graphics.hpp>
5  #include "TFractal.h"
6  #include "Triangle.h"
7
8  int main(int argc, char* argv[])
9  {
10     if(argc!=3){
11         std::cout << "input length of triangle and number of iteration";
12         return -1;
13     }
14
15     std::string length(argv[1]);
16     std::string iteration(argv[2]);
17
18     double Length = std::atoi(length.c_str());
19     int width{(int)Length * 6};
20     int height{(int)Length * 4};
21     double Iteration = std::atoi(iteration.c_str());
22
23     sf::RenderWindow window(sf::VideoMode(width, height), "Window");
24     TFractal *obj = new TFractal();
25     obj->fTree(Iteration, Length, width, height);
26     Triangle* triangle5= obj->triangle2;
27
28     while(window.isOpen()){
29         sf::Event event;
30         while(window.pollEvent(event)){
31             if(event.type == sf::Event::Closed){
32                 window.close();
33             }
34             else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)){
35                 window.close();
36             }
37         }
38         window.clear();
39         window.draw(*triangle5);
40         window.display();
41     }
42
43     return 0;
44 }

```

```
1  #ifndef TFractal_H
2  #define TFractal_H
3
4  #include "../Triangle.h"
5  #include <string>
6  #include <SFML/Graphics.hpp>
7
8  class TFractal{
9      public:
10         void fTree(int depth, double length, int WindowWidth, int WindowHeight);
11         Triangle *triangle2;
12         Triangle *fTree(Triangle* triangle3, int depth);
13
14         void setChildPosition(Triangle *triangle3);
15
16     };
17 #endif
18 //TFractal_H
```

```

1  #include <iostream>
2  #include <stdexcept>
3  #include <sstream>
4  #include <SFML/Graphics.hpp>
5  #include "TFractal.h"
6  #include "Triangle.h"
7
8  void TFractal::fTree(int depth, double length, int WindowWidth, int WindowHeight){
9      triangle2= new Triangle(length);
10     triangle2-> move ((WindowWidth/2) - (length/2),(WindowHeight/2)-(length/2));
11     triangle2->addLeftChild(fTree(triangle2,depth));
12     triangle2->addRightChild(fTree(triangle2,depth));
13     triangle2->addUnderChild(fTree(triangle2,depth));
14
15     setChildPosition(triangle2);
16 }
17
18 Triangle* TFractal::fTree(Triangle *triangle3, int depth){
19     if(depth<=0){
20         return nullptr;
21     }
22
23     double baseLength = triangle3->length;
24
25     Triangle *triangleChild = new Triangle(baseLength/2);
26     triangleChild->addLeftChild(fTree(triangleChild, depth-1));
27     triangleChild->addRightChild(fTree(triangleChild, depth-1));
28     triangleChild->addUnderChild(fTree(triangleChild, depth-1));
29
30     return triangleChild;
31 }
32
33 void TFractal::setChildPosition(Triangle* baseTriangle){
34     if(baseTriangle == nullptr || baseTriangle->triangleLeft==nullptr){
35         return;
36     }
37
38     sf::Vector2f basePosition{ baseTriangle->getPosition()};
39     double baseLength{ baseTriangle->length};
40     double childLength{ baseLength/2};
41
42     baseTriangle->triangleLeft->move(basePosition.x-childLength/2, basePosition.y-childLength);
43     baseTriangle->triangleRight->move(basePosition.x + baseLength, basePosition.y);
44     baseTriangle->triangleUnder->move(basePosition.x, basePosition.y+baseLength);
45
46     setChildPosition(baseTriangle->triangleRight);
47     setChildPosition(baseTriangle->triangleLeft);
48     setChildPosition(baseTriangle->triangleUnder);
49 }
50

```

```

1  #ifndef Triangle_H
2  #define Triangle_H
3
4  #include <string>
5  #include <SFML/Graphics.hpp>
6
7  class Triangle : public sf::Drawable
8  {
9      public:
10         Triangle(double);
11         void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;
12         double length;
13
14         sf::ConvexShape triangle2;
15
16         Triangle *triangleLeft;
17         Triangle *triangleRight;
18         Triangle *triangleUnder;
19
20         void move(int x, int y);
21         sf::Vector2f getPosition();
22
23         void setOrigin(sf::Vector2f);
24         void setFill(sf::Color);
25         void addLeftChild(Triangle *triangle2);
26         void addRightChild(Triangle *triangle2);
27         void addUnderChild(Triangle *triangle2);
28
29         Triangle* getLeft();
30         Triangle* getRight();
31         Triangle* getUnder();
32
33     };
34 #endif
35 //Triangle_H

```

```

1  #include "Triangle.h"
2
3  Triangle::Triangle(double length)
4  {
5      triangle2 = sf::ConvexShape();
6      triangle2.setPointCount(3);
7      triangle2.setPoint(0, sf::Vector2f(0,0));
8      triangle2.setPoint(1, sf::Vector2f(length,0));
9      triangle2.setPoint(2, sf::Vector2f(length/2, length));
10     this->length = length;
11
12     return;
13
14 }
15
16 //mutator functions
17 void Triangle::move(int x, int y){
18     triangle2.move(sf::Vector2f(x,y));
19 }
20
21 sf::Vector2f Triangle::getPosition(){
22     return triangle2.getPosition();
23 }
24
25 void Triangle::setOrigin(sf::Vector2f origin){
26     this->triangle2.setOrigin(origin);
27 }
28
29 void Triangle::setFill(sf::Color color){
30     triangle2.setFillColor(color);
31 }
32
33 void Triangle::addLeftChild(Triangle *triangleL){
34     this->triangleLeft = triangleL;
35 }
36
37 void Triangle::addRightChild(Triangle *triangleR){
38     this->triangleRight = triangleR;
39 }
40
41 void Triangle::addUnderChild(Triangle *triangleU){
42     this->triangleUnder = triangleU;
43 }
44
45 Triangle* Triangle::getLeft(){
46     return triangleLeft;
47 }
48
49 Triangle* Triangle::getRight(){
50     return triangleRight;
51 }
52
53 Triangle* Triangle::getUnder(){
54     return triangleUnder;
55 }
56
57 void Triangle::draw(sf::RenderTarget& target, sf::RenderStates states)const{
58     target.draw(triangle2);
59     if(this->triangleUnder != nullptr) target.draw(*triangleUnder);
60     if(this->triangleRight != nullptr) target.draw(*triangleRight);
61     if(this->triangleLeft != nullptr) target.draw(*triangleLeft);
62
63
64 }
65

```

PS4: Synthesizing a Plucked String Sound

Part A: CircularBuffer Implementation

The assignment 4 is about using the functions of queue to implement Circular Buffer. The Circular Buffer class includes creating empty ring, returning size, checking if it's empty or full, inserting, removing and peeking items. Each function also implements runtime and invalid argument error exceptions when the queue is added more items when it is full, or an item is removed from the queue when it is empty. This assignment uses Boost test framework again to test out if each exception implementation is working or not. I tested three cases- the first one tests that the Circular Buffer object should not be created 0, the second one tests if the queue is empty or full, and the final case tests if the runtime error on adding new items when the queue is full and on peek function. From this assignment, I learned more about the boost test and exceptions.

```
BOOST_AUTO_TEST_CASE(test1){
    BOOST_REQUIRE_THROW(CircularBuffer c(0), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(test2){
    CircularBuffer c(12);
    BOOST_REQUIRE(c.isEmpty() == true);
    BOOST_REQUIRE(c.isFull() == false);
}

BOOST_AUTO_TEST_CASE(test3){
    CircularBuffer c(3);
    c.enqueue(1);
    c.enqueue(2);
    c.enqueue(3);
    BOOST_REQUIRE_THROW(c.enqueue(4), std::runtime_error);
    BOOST_REQUIRE(c.dequeue() == 1);
    BOOST_REQUIRE_NO_THROW(c.peek() == 2);
    BOOST_REQUIRE(c.dequeue() == 2);
    BOOST_REQUIRE(c.dequeue() == 3);
    BOOST_REQUIRE_THROW(c.peek(), std::runtime_error);
}
```

Part B: StringSound Implementation

The part B of assignment added new StringSound class to implement the Karplus-Strong Algorithm. The main function of the program is to produce sound based on the key pressed on keyboard with corresponding key notes. The frequencies were calculated as $440 * 2^{(i-24)/12}$ where i^{th} character of the keyboard. There are 37 key notes to represent 37 keys on keyboard. The StringSound class has pluck, tic, sample, and time functions and also pointers to retrieve or populate data in CircularBuffer. The

main file uses SFML functions- sf::sound, sf::soundbuffer and sf::int16 as well as event.text.unicode and TextEntered to check if buttons are pressed and to test the keys. The vector is first created and that same vector sample is passed through soundbuffer and the result is added into sound vector which plays different key notes with the corresponding key pressed on keyboard. The program is just an empty window which plays different key notes. From this assignment, I learned more about SFML functions.

Implementation of Karplus-Strong Algorithm

```
int notes = 37;
std::vector<sf::SoundBuffer> sb(notes);
std::vector<sf::Sound> sd(notes);
std::vector<std::vector<sf::Int16> > samples(notes);

for(int i = 0; i<notes; i++){
    freq = 440 * pow(2, (i-24)/12.0);
    StringSound gs = StringSound(freq);
    samples.at(i) = makeSamplesFromString(gs);
    sb.at(i).loadFromSamples(&samples.at(i)[0], samples.at(i).size(), 2, SAMPLES_PER_SEC);
    sd.at(i).setBuffer(sb.at(i));
}
```

Files:

Makefile

Circularbuffer.h

Circularbuffer.cpp

StringSound.h

StringSound.cpp

KSGuitarSim.cpp

```

1  CC = g++
2  CFLAGS = -Wall -Werror -ansi --pedantic -g -lsfml-audio -lsfml-graphics -lsfml-window -lsfml-system
3  OBJ = CircularBuffer.o KSGuitarSim.o StringSound.o
4  DEPS = CircularBuffer.h StringSound.h
5  LIBS =
6  EXE = StringSound
7
8
9  all: $(OBJ)
10     $(CC) $(OBJ) -o $(EXE) $(LIBS) $(CFLAGS)
11
12  KSGuitarSim.o: KSGuitarSim.cpp
13     $(CC) -c KSGuitarSim.cpp
14
15  CircularBuffer.o: CircularBuffer.cpp
16     $(CC) -c CircularBuffer.cpp
17
18  StringSound.o: StringSound.cpp
19     $(CC) -c StringSound.cpp
20
21
22  clean:
23     rm $(OBJ) $(EXE)

```

```

1  #ifndef CIRCULARBUFFER_H
2  #define CIRCULARBUFFER_H
3
4  #include <stdint.h>
5  #include <vector>
6  #include <stdexcept>
7  #include <iostream>
8
9  class CircularBuffer{
10 public:
11     CircularBuffer(int capacity);
12     ~CircularBuffer();
13     void KarplusStrong();
14     int size();
15     bool isEmpty();
16     bool isFull();
17     void enqueue(int16_t x);
18     int16_t dequeue();
19     int16_t peek();
20     friend std::ostream &operator <<(std::ostream &out, const CircularBuffer &C);
21
22 private:
23     int cap;
24     std::vector<int16_t> vect;
25 };
26
27 #endif

```

```

1  #include "CircularBuffer.h"
2  #include <vector>
3
4  int CircularBuffer::size(){
5      return vect.size();
6  }
7
8  bool CircularBuffer::isEmpty(){
9      if(vect.empty())
10         return true;
11     else
12         return false;
13 }
14
15 CircularBuffer::CircularBuffer(int capacity){
16     if(capacity == 0){
17         throw std::invalid_argument("CircularBuffer constructor: capacity must be greater than zero.");
18     }
19     cap = capacity;
20 }
21
22
23 bool CircularBuffer::isFull(){
24     if(vect.size() == cap)
25         return true;
26     else
27         return false;
28 }
29
30 void CircularBuffer::enqueue(int16_t x){
31     if (isFull()){
32         throw std::runtime_error("enqueue: can't enqueue to a full ring.");
33     }
34     vect.push_back(x);
35 }
36
37 int16_t CircularBuffer::dequeue(){
38     int16_t t = peek();
39     vect.erase(vect.begin());
40     return t;
41 }
42
43 int16_t CircularBuffer::peek(){
44     if (isEmpty()){
45         throw std::runtime_error("CircularBuffer is empty.");
46     }
47     return vect.at(0);
48 }
49
50 void CircularBuffer::KarplusStrong(){
51     enqueue((.996 * 1/2) * (vect.at(0)+vect.at(1)));
52     dequeue();
53     return;
54 }
55
56 std::ostream & operator << (std::ostream &out, const CircularBuffer &C){
57     for (int i = 0; i < C.vect.size(); ++i){
58         out << i << " " << C.vect.at(i) << std::endl;
59     }
60     return out;
61 }

```

```
1 #include <SFML/Graphics.hpp>
2 #include <SFML/System.hpp>
3 #include <SFML/Audio.hpp>
4 #include <vector>
5 #include "CircularBuffer.h"
6
7 class StringSound {
8     public:
9         explicit StringSound(double frequency);
10        explicit StringSound(std::vector<sf::Int16> init);
11        ~StringSound();
12        void pluck();
13        void tic();
14        sf::Int16 sample();
15        int time();
16    private:
17        CircularBuffer* CB;
18        int numOfTic;
19 };
```

```

1  #include "StringSound.h"
2  #include "CircularBuffer.h"
3  #include <SFML/Graphics.hpp>
4  #include <SFML/System.hpp>
5  #include <SFML/Audio.hpp>
6  #include <SFML/Window.hpp>
7  #include <math.h>
8  #include <vector>
9  #include <stdexcept>
10 #include <exception>
11
12 StringSound::StringSound(double frequency){
13     numOfTic = 0;
14     if(frequency <= 0){
15         throw std::invalid_argument("Frequency must be positive number");
16     }
17     else {
18         CB = new CircularBuffer(ceil(44100 / frequency));
19         while (!CB->isFull()){
20             CB->enqueue(0);
21         }
22     }
23 }
24
25 StringSound::StringSound(std::vector<sf::Int16> init){
26     numOfTic = 0;
27     if (init.size() <= 0){
28         throw std::invalid_argument("Size must be positive numver");
29     }
30     else{
31         CB = new CircularBuffer(init.size());
32         int i = 0;
33         while (!CB->isFull()){
34             CB->enqueue(init[i]);
35             i++;
36         }
37     }
38 }
39
40 StringSound::~StringSound(){
41 }
42
43 void StringSound::pluck(){
44     if(CB->isFull()){
45         while(!CB->isEmpty()){
46             CB->dequeue();
47         }
48     }
49     while(!CB->isFull()){
50         CB->enqueue((rand()% INT16_MAX)+ 1);
51     }
52 }
53
54 void StringSound::tic(){
55     CB->enqueue(0.5 * 0.996 * (CB->dequeue() + CB->peek()));
56     numOfTic++;
57     if (numOfTic == 100 && CB->size() >= 100){
58         std::vector<sf::Int16> queue;
59         int i = 0;
60         while (i<100 && !CB->isEmpty()){
61             queue.push_back(CB->dequeue());
62             i++;
63         }
64         i = 0;
65         while (i < 99 && !CB->isFull()){
66             CB->enqueue(queue.at(i));
67             i++;
68         }
69     }
70 }
71
72 sf::Int16 StringSound::sample(){
73     return CB->peek();
74 }
75
76 int StringSound::time(){
77     return numOfTic;
78 }

```

```

1  #include <SFML/Graphics.hpp>
2  #include <SFML/System.hpp>
3  #include <SFML/Audio.hpp>
4  #include <SFML/Window.hpp>
5  #include <math.h>
6  #include <limits.h>
7  #include <iostream>
8  #include <string>
9  #include <exception>
10 #include <stdexcept>
11 #include <vector>
12 #include "CircularBuffer.h"
13 #include "StringSound.h"
14
15 #define CONCERT_A 220.0
16 #define SAMPLES_PER_SEC 44100
17
18 std::vector<sf::Int16> makeSamplesFromString(StringSound gs) {
19     std::vector<sf::Int16> samples;
20
21     gs.pluck();
22     int duration = 8;
23     int i;
24     for (i = 0; i < SAMPLES_PER_SEC * duration; i++){
25         gs.tic();
26         samples.push_back(gs.sample());
27     }
28     return samples;
29 }
30
31 int main(){
32     sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Plucked String Sound Lite");
33     sf::Event event;
34     double freq;
35     int notes = 37;
36     std::vector<sf::SoundBuffer> sb(notes);
37     std::vector<sf::Sound> sd(notes);
38     std::vector<std::vector<sf::Int16> > samples(notes);
39
40     for(int i = 0; i < notes; i++){
41         freq = 440 * pow(2, (i-24)/12.0);
42         StringSound gs = StringSound(freq);
43         samples.at(i) = makeSamplesFromString(gs);
44         sb.at(i).loadFromSamples(&samples.at(i)[0], samples.at(i).size(), 2, SAMPLES_PER_SEC);
45         sd.at(i).setBuffer(sb.at(i));
46     }
47
48     std::cout << "Loading is done" << std::endl;
49     std::string keys = "q2we4r5ty7u8i9op-=[zxdcfvgbnjmk,./' ";
50     while (window.isOpen()){
51         while(window.pollEvent(event)){
52             if(event.type == sf::Event::Closed)
53                 window.close();
54
55             for (int i = 0; i < 37; i++){
56                 if (event.type == sf::Event::TextEntered){
57                     if(event.text.unicode == keys.at(i)){
58                         sd.at(i).play();
59                     }
60                 }
61             }
62             window.clear();
63             window.display();
64         }
65     }
66     return 0;

```

PS5: DNA Sequence Alignment Ring

The assignment PS5 is to compute the optimal sequence of alignment of 2 DNA strings and learn about dynamic programming. Dynamic Programming is about breaking down large components of a problem into smaller ones as subproblems and solve them first before solving the problem as a whole. This idea is used in this assignment to find the edit distance as well as Needleman and Wunsch algorithm. The 2 strings are placed in a 2x2 dimension array and the edit distance is calculated based on the penalties of the strings (return 0 if matched, 1 if not matched). The min function chooses 3 arguments- down, right, and cross. Gap (-) always add 2 to the costs so down, and right will always add 2 to the costs. Cross, however, uses the penalty value from penalty function. If the characters are matched, add 0 to costs, and if they are unmatched, add 1 to the costs. In this way, the edit distance is calculated until at the final position [0][0]. The two functions OptDistance and Alignment return the data. The OptDistance function retruns the edit distance value and the Alignment function traces back the matrix and returns the string alignment of 2 DNA sequences. Below shows the example 2x2 dimension array of 2 DNA sequences:

x\y		0	1	2	3	4	5	6	7	8
		T	A	A	G	G	T	C	A	-
0	A	7	8	10	12	13	15	16	18	20
1	A	6	6	8	10	11	13	14	16	18
2	C	6	5	6	8	9	11	12	14	16
3	A	7	5	4	6	7	9	11	12	14
4	G	9	7	5	4	5	7	9	10	12
5	T	8	8	6	4	4	5	7	8	10
6	T	9	8	7	5	3	3	5	6	8
7	A	11	9	7	6	4	2	3	4	6
8	C	13	11	9	7	5	3	1	3	4
9	C	14	12	10	8	6	4	2	1	2
10	-	16	14	12	10	8	6	4	2	0

This is the table for costs of the operation.

operation	cost
insert a gap	2
align two characters that mismatch	1
align two characters that match	0

The program takes in a .txt input file which includes the 2 DNA sequences. The assignment requires to test multiple .txt files to determine the analysis of the dynamic programming efficiency. For very large strings of data, the memory and time taken may vary. For example, for 2500 characters, the edit distance is 118 with 0.07127s and 24.02 MB of memory. For the largest file with 28284 characters, the edit distance is 8394 with 10.14s and 2981 MB of memory. I use valgrind to analyze each run of strings and produce the analyzed data. From this assignment, I learned about dynamic programming, analysis of efficiency of programs using valgrind.

Files:

Makefile

EDistance.h

EDistance.cpp

main.cpp

```

1 CC = g++
2 CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic -lsfml-graphics -lsfml-window -lsfml-system
3 OBJS = EDistance.o main.o
4 DEPS = EDistance.h
5 LIBS =
6 EXE = EDistance
7
8 all: $(OBJS)
9     $(CC) $(OBJS) -o $(EXE) $(LIBS) $(CFLAGS)
10
11 main.o: main.cpp
12     $(CC) -c main.cpp
13
14 EDistance.o: EDistance.cpp EDistance.h
15     $(CC) -c EDistance.cpp
16
17 clean:
18     rm $(OBJS) $(EXE)

```

```

1 #ifndef EDISTANCE_H
2 #define EDISTANCE_H
3 #include <string>
4
5 class EDistance{
6     public:
7         EDistance(std::string, std::string);
8         ~EDistance();
9         static int penalty(char a, char b);
10        static int min(int a, int b, int c);
11        int optDistance();
12        std::string alignment();
13        friend std::ostream& operator<<(std::ostream&, EDistance&);
14
15    private:
16        int** opt;
17        std::string xString, yString;
18 };
19
20 #endif

```

```

1  #include "EDistance.h"
2  #include <iostream>
3
4  EDistance::EDistance(std::string x, std::string y){
5      xString = x + "-";
6      yString = y + "-";
7
8      opt = new int*[xString.length()];
9      for (int i = 0; i < xString.length(); ++i){
10         opt[i] = new int[yString.length()];
11     }
12 }
13
14 EDistance::~EDistance(){
15     for(int i = 0; i<xString.length(); ++i){
16         delete [] opt[i];
17     }
18     delete [] opt;
19 }
20
21 int EDistance::penalty(char x, char y){
22     return (x!=y);
23 }
24
25 int EDistance::min(int down, int right, int diagonal){
26     return std::min(std::min(down, right), diagonal);
27 }
28
29 int EDistance::optDistance(){
30     int num = 0;
31     for(int i = xString.length()-1; i>=0; i--){
32         opt[i][yString.length()-1] = num * 2;
33         num++;
34     }
35     num = 0;
36     for(int j = yString.length()-1; j>=0; j--){
37         opt[xString.length()-1][j] = num * 2;
38         num++;
39     }
40
41     for(int i = xString.length()-1; i > 0; i--){
42         for(int j = yString.length()-1; j>0; j--){
43             int minNum = min(opt[i][j-1] + 2, opt[i-1][j] + 2, opt[i][j] + (penalty(xString.at(i-1), yString.at(j-1))));
44             opt[i-1][j-1] = minNum;
45         }
46     }
47     return opt[0][0];
48 }
49
50 std::string EDistance::alignment(){
51     return std::string();
52 }
53
54 std::ostream &operator << (std::ostream &os, EDistance &ed){
55     os << ed.xString << ed.yString;
56     return os;
57 }
58

```

```
1 #include <iostream>
2 #include "EDistance.h"
3 #include <SFML/System.hpp>
4
5 int main(){
6     sf::Clock clock;
7     sf::Time t;
8
9     std::string x,y;
10    std::cin >> x >> y;
11
12    EDistance ed(x,y);
13
14    std::cout << "Edit Distance: " << ed.optDistance() << std::endl;
15    //std::cout << ed << std::endl;
16    t = clock.getElapsedTime();
17
18    std::cout << "Execution time is " << t.asSeconds() << " seconds \n";
19
20 }
```

PS6: Markov Model of Natural Language

The assignment PS6 is based on Markov Model which is the work of Claude Shannon. The model predicts the probability of each letter of alphabet appearing in word. The program takes in a parse file in .txt form, the order of K and the length of output (L). The order of K is the length of K strings stored as kgram, and to predict the probability, the frequency of kgram is recorded. That frequency is implemented in function freq and returns the number of times kgram appeared in the given string. The values of kgram are stored in multimap which requires every element to have keys. The function kRand takes in kgram as argument and returns the highest chance of next letter/character appearing after that kgram. Those characters are stored in a vector. Lastly, the generate functions returns the string of the length of the output. This assignment tests cases with boost test framework.

This is how kgrms are stored in multimap.

```
for(int i= 0; i<=inputText.length()-k -1; i++){
    for (auto &key : kgramMap){
        if (key.first == inputText.substr(i, k) && key.second.first == inputText.at(i + k)){
            key.second.second++;
            found = true;
        }
    }
    if(!found ){
        kgramMap.insert({inputText.substr(i,k), {inputText.at(i+k), 1}});
    }
    found = false;
}
```

The test.cpp tests all the functions of the program. The program takes in a string and the order K. If the string is less than the order, it will run invalid argument error. The function orderK returns the order of the string, the freq returns the frequencies of kgram and the kRand predicts the highest chance of characters appearing after the given kgram. From this assignment, I learned about using multimaps and more about boost testing functions.

```

BOOST_AUTO_TEST_CASE(Constructor){
    BOOST_REQUIRE_NO_THROW(RandWriter("gagggagagggcgagaaa", 2));
    BOOST_REQUIRE_THROW(RandWriter("", 3), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(kOrder){
    RandWriter mm("gagggagagggcgagaaa", 2);
    RandWriter m2("gagggagagggcgagaaa", 5);

    BOOST_REQUIRE(mm.orderK() == 2);
    BOOST_REQUIRE(m2.orderK() == 5);
}

BOOST_AUTO_TEST_CASE(freq){
    RandWriter mm("gagggagagggcgagaaa", 2);

    BOOST_REQUIRE(mm.freq("ga") == 5);
    BOOST_REQUIRE(mm.freq("aa") == 2);

    BOOST_REQUIRE_THROW(mm.freq("www"), std::runtime_error);
    BOOST_REQUIRE_THROW(mm.freq("rwv"), std::runtime_error);

    BOOST_REQUIRE_THROW(mm.freq("www", 'c') != 1, std::runtime_error);
    BOOST_REQUIRE_THROW(mm.freq("gag", 'c') != 1, std::runtime_error);
}

BOOST_AUTO_TEST_CASE(kRand){
    RandWriter mm("gagggagagggagagagagagagagagagagagggcgagaaa", 2);
    RandWriter m2("gaaaagaaaagaaaag", 5);

    BOOST_REQUIRE(mm.kRand("ga") == 'g');
    BOOST_REQUIRE(m2.kRand("gaaaa") == 'g');

    BOOST_REQUIRE_THROW(mm.kRand("ggg") == 'a', std::runtime_error);
    BOOST_REQUIRE_THROW(m2.kRand("burp") != 'g', std::runtime_error);
}

BOOST_AUTO_TEST_CASE(OTHERS){
    RandWriter mm("gagggagagggcgagaaa", 2);
    BOOST_REQUIRE(mm.orderK() == 2);
}

```

Files:

Makefile

RandWriter.h

RandWriter.cpp

test.cpp

TextWriter.cpp

```
1 CC = g++
2 CFLAGS = -Wall -Werror -ansi --pedantic -g
3 OBJS = RandWriter.o test.o
4 DEPS = RandWriter.h
5 LIBS = -lboost_unit_test_framework
6 EXE = TextWriter
7
8 all: $(OBJS)
9     $(CC) $(OBJS) -o $(EXE) $(LIBS) $(CFLAGS)
10
11 test.o: test.cpp
12     $(CC) -c test.cpp
13
14 RandWriter.o: RandWriter.cpp RandWriter.h
15     $(CC) -c RandWriter.cpp
16
17 clean:
18     rm $(OBJS) $(EXE)
```

```

1  #ifndef PS6_RandWriter_H_
2  #define PS6_RandWriter_H_
3
4  #include <iostream>
5  #include <map>
6  #include <iterator>
7  #include <algorithm>
8  #include <ctime>
9  #include <vector>
10 #include <string>
11 #include <utility>
12 #include <memory>
13 #include <fstream>
14
15 class RandWriter {
16     private:
17         std::string inputText, outputText, text;
18         int k, outputLength;
19
20         std::multimap<std::string, std::pair<char, int> > kgramMap;
21
22     public:
23         RandWriter(std::string, int);
24         int orderK();
25         int freq(std::string, char);
26         int freq(std::string);
27         char kRand(std::string);
28         std::string generate(std::string, int);
29
30         friend std::ostream& operator << (std::ostream&, RandWriter&);
31 };
32
33 #endif

```

```

1  #include "RandWriter.h"
2
3  RandWriter::RandWriter(std::string text, int k){
4      std::srand(time(NULL));
5
6      if (text.length() == 0 || k < 0){
7          throw std::invalid_argument("Empty String");
8      }
9      this->text = text;
10     inputText = text;
11     this->k = k;
12     bool found = false;
13     inputText += inputText.substr(0, k);
14
15     for(int i = 0; i <= inputText.length() - k - 1; i++){
16         for (auto &key : kgramMap){
17             if (key.first == inputText.substr(i, k) && key.second.first == inputText.at(i + k)){
18                 key.second.second++;
19                 found = true;
20             }
21         }
22         if(!found){
23             kgramMap.insert({inputText.substr(i,k), {inputText.at(i+k), 1}});
24         }
25         found = false;
26     }
27 }
28
29 int RandWriter::orderK(){
30     return k;
31 }
32
33 int RandWriter::freq(std::string k_gram, char c){
34     int charFrequency = 0;
35
36     if(k_gram.length() != k){
37         throw std::runtime_error("Kgram is less or greater than order k");
38     }
39
40     if(k == 0){
41         for (int i = 0; i < inputText.length() - 1; ++i){
42             if(inputText.at(i) == c){
43                 charFrequency++;
44             }
45         }
46     }
47     else {
48         for (auto &key : kgramMap){
49             if (key.first == k_gram){
50                 if(c == key.second.first){
51                     charFrequency = key.second.second;
52                 }
53             }
54         }
55     }
56     return charFrequency;
57 }
58
59 int RandWriter::freq(std::string k_gram){
60     int frequency = 0;
61     if (k_gram.length() != k){
62         throw std::runtime_error("kgram is less than or greater than order k");
63     }
64     else {
65         for (auto &key : kgramMap){
66             if (key.first == k_gram){
67                 frequency += key.second.second;
68             }
69         }
70     }
71     return frequency++;
72 }

```



```

73
74 char RandWriter::kRand(std::string k_gram){
75     std::vector<char> charFreq;
76
77     if (k_gram.length() != k){
78         throw std::runtime_error("kgram is not length of k");
79     }
80
81     auto iter = kgramMap.find(k_gram);
82
83     if(iter == kgramMap.end()){
84         throw std::runtime_error("No kgram found in given text");
85     }
86
87     for (auto &key: kgramMap){
88         if(key.first == k_gram){
89             for (int i = 0; i < key.second.second; ++i){
90                 charFreq.push_back(key.second.first);
91             }
92         }
93     }
94     return charFreq.at(rand() % charFreq.size());
95 }
96
97
98 std::string RandWriter::generate(std::string k_gram, int L){
99     std::string outputText = k_gram;
100     std::string temp = k_gram;
101     char chartemp;
102     for (int i = 0; i < L - k_gram.length(); ++i){
103         chartemp = kRand(temp);
104         outputText += chartemp;
105         temp = temp.substr(1,k) + chartemp;
106     }
107     return outputText;
108 }
109
110 std::ostream& operator << (std::ostream& output, RandWriter &karnov){
111     output << "Order of: " << karnov.k << "\n";
112     output << "    Input: " << karnov.text << "\n";
113     //output << "    Output: " << karnov.generate(karnov.inputText.substr(0, karnov.k), karnov.output(Length));
114     for(auto &key : karnov.kgramMap){
115         output << key.first << "Freq:" << karnov.freq(key.first) << "Next character: "
116             << key.second.first << "with freq of: " << key.second.second << std::endl;
117     }
118     return output;
119 }

```

```

1  #define BOOST_TEST_DYN_LINK
2  #define BOOST_TEST_MODULE Main
3
4  #include <stdint.h>
5  #include <iostream>
6  #include <string>
7  #include <exception>
8  #include <stdexcept>
9  #include <boost/test/unit_test.hpp>
10 #include "RandWriter.h"
11
12 BOOST_AUTO_TEST_CASE(Constructor){
13     BOOST_REQUIRE_NO_THROW(RandWriter("gagggagaggcgagaaa", 2));
14     BOOST_REQUIRE_THROW(RandWriter("", 3), std::invalid_argument);
15 }
16
17 BOOST_AUTO_TEST_CASE(kOrder){
18     RandWriter mm("gagggagaggcgagaaa", 2);
19     RandWriter m2("gagggagaggcgagaaa", 5);
20
21     BOOST_REQUIRE(mm.orderK() == 2);
22     BOOST_REQUIRE(m2.orderK() == 5);
23
24 }
25
26 BOOST_AUTO_TEST_CASE(freq){
27     RandWriter mm("gagggagaggcgagaaa", 2);
28
29     BOOST_REQUIRE(mm.freq("ga") == 5);
30     BOOST_REQUIRE(mm.freq("aa") == 2);
31
32     BOOST_REQUIRE_THROW(mm.freq("www"), std::runtime_error);
33     BOOST_REQUIRE_THROW(mm.freq("rwv"), std::runtime_error);
34
35     BOOST_REQUIRE_THROW(mm.freq("www", 'c') != 1, std::runtime_error);
36     BOOST_REQUIRE_THROW(mm.freq("gag", 'c') != 1, std::runtime_error);
37 }
38
39 BOOST_AUTO_TEST_CASE(kRand){
40     RandWriter mm("gagggagaggagagagagagagagaggagaggcgagaaa", 2);
41     RandWriter m2("gaaaagaaaagaaaag", 5);
42
43     BOOST_REQUIRE(mm.kRand("ga") == 'g');
44     BOOST_REQUIRE(m2.kRand("gaaaa") == 'g');
45
46     BOOST_REQUIRE_THROW(mm.kRand("ggg") == 'a', std::runtime_error);
47     BOOST_REQUIRE_THROW(m2.kRand("burp") != 'g', std::runtime_error);
48 }
49
50 BOOST_AUTO_TEST_CASE(OTHERS){
51     RandWriter mm("gagggagaggcgagaaa", 2);
52     BOOST_REQUIRE(mm.orderK() == 2);
53 }

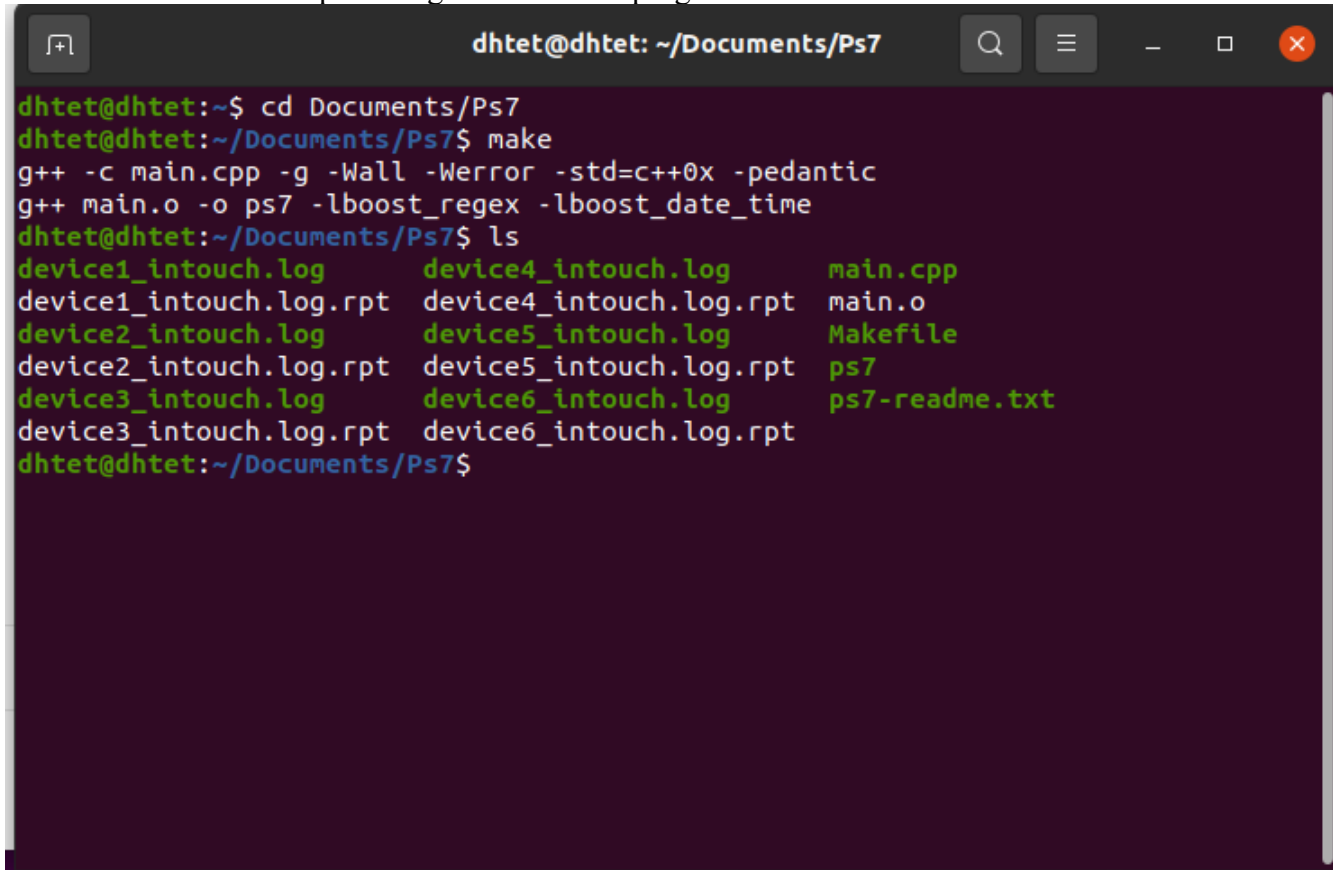
```

```
1 #include <iostream>
2 #include "RandWriter.h"
3
4 int main(){
5     RandWriter km("gaagggagaggaga", 2);
6
7     std::cout << km.kOrder() << std::endl;
8     km.k_Rand("ta");
9 }
```

PS7: Kronos Time Clock or Introduction to Regular Expression Parsing

The assignment PS7 is about the Kronos InTouch device which is a Linux-based touch screen time clocks that integrates with different host-based systems. The Kronos Service and Engineering gathers information when error occurs on the device at different times of failures. They use the InTouch log files to gather error information and when they are retrieved, the device gets a review for error. This assignment is about production those log files with specific data information. From this assignment, I learned about the regular expression which is also known as regex which helps to find characters or sequences with a specific syntax.

These are the output of log files from the program.

A terminal window titled 'dhtet@dhtet: ~/Documents/Ps7' with standard window controls. The terminal shows the following commands and output:

```
dhtet@dhtet:~$ cd Documents/Ps7
dhtet@dhtet:~/Documents/Ps7$ make
g++ -c main.cpp -g -Wall -Werror -std=c++0x -pedantic
g++ main.o -o ps7 -lboost_regex -lboost_date_time
dhtet@dhtet:~/Documents/Ps7$ ls
device1_intouch.log      device4_intouch.log      main.cpp
device1_intouch.log.rpt  device4_intouch.log.rpt  main.o
device2_intouch.log      device5_intouch.log      Makefile
device2_intouch.log.rpt  device5_intouch.log.rpt  ps7
device3_intouch.log      device6_intouch.log      ps7-readme.txt
device3_intouch.log.rpt  device6_intouch.log.rpt
dhtet@dhtet:~/Documents/Ps7$
```

Files:

Makefile

main.cpp

```
1 CC = g++
2 CFLAGS = -g -Wall -Werror -std=c++0x -pedantic
3 SFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
4 Boost = -lboost_regex -lboost_date_time
5
6 all:    ps7
7
8 ps7:    main.o
9         $(CC) main.o -o ps7 $(Boost)
10
11 main.o: main.cpp
12         $(CC) -c main.cpp $(CFLAGS)
13
14 clean:
15         rm *.o
16         rm ps7
```

```

1  #include <boost/regex.hpp>
2  #include <iostream>
3  #include <fstream>
4  #include <string>
5  #include "boost/date_time/gregorian/gregorian.hpp"
6  #include "boost/date_time/posix_time/posix_time.hpp"
7
8  using boost::gregorian::date;
9  using boost::gregorian::from_simple_string;
10 using boost::gregorian::date_period;
11 using boost::gregorian::date_duration;
12
13 using boost::posix_time::ptime;
14 using boost::posix_time::time_duration;
15
16 int main(int argc, const char* argv[]){
17     if(argc != 2){
18         std::cout << "./ps7 device1_intouch.log\n";
19         return 0;
20     }
21
22     int lines_scan = 1;
23     int boot_success = 0;
24     int boot_total = 0;
25
26     std::string file_name(argv[1]);
27     std::string output_name = file_name + ".rpt";
28     std::string report = "";
29     std::string boots = "";
30
31     std::string begin_date = "";
32     std::string end_date = "";
33
34     std::string full_date;
35     int hours = 0;
36     int minutes = 0;
37     int seconds = 0;
38
39     ptime begin;
40     ptime end;
41
42     date date1;
43     date date2;
44
45     time_duration time_diff;
46
47     std::string start_string = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
48     start_string += "([0-9]{2}):([0-9]{2}):([0-9]{2}): \\(log.c.166\\) ";
49     start_string += "sever started";
50     std::string end_string = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
51     end_string += "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}):INFO:oejs.";
52     end_string += "AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080";
53
54     boost::regex start_regex(start_string, boost::regex::perl);
55     boost::regex end_regex(end_string);
56
57     boost::smatch sm;
58
59     std::string line;
60     std::ifstream file(file_name.c_str());
61
62     bool found_start = false;

```

```

64 if(file.is_open()){
65     while (getline(file, line)){
66         begin_date.clear();
67         end_date.clear();
68
69         if(boost::regex_search(line, sm, start_regex)){
70             begin_date = sm[1] + "-" + sm[2] + "-" + sm[3];
71             begin_date += " " + sm[4] + ":" + sm[5] + ":" + sm[6];
72
73             full_date = sm[1] + "-" + sm[2] + "-" + sm[3];
74             date1 = date(from_simple_string(full_date));
75
76             hours = std::stoi(sm[4]);
77             minutes = std::stoi(sm[5]);
78             seconds = std::stoi(sm[6]);
79
80             begin = ptm(date1, time_duration(hours, minutes, seconds));
81
82             if(found_start == true){
83                 boots += "**** Incomplete boot **** \n\n";
84             }
85
86             boots += "=== Device boot ===\n";
87             boots += std::to_string(lines_scan) + "(" + file_name + "): ";
88             boots += begin_date + "Boot Start\n";
89
90             boot_total++;
91             found_start = true;
92         }
93         if (boost::regex_match(line, sm, end_regex)){
94             end_date = sm[1] + "-" + sm[2] + "-" + sm[3];
95             end_date += " " + sm[4] + ":" + sm[5] + ":" + sm[6];
96
97             full_date = sm[1] + "-" + sm[2] + "-" + sm[3];
98             date2 = date(from_simple_string(full_date));
99
100             hours = std::stoi(sm[4]);
101             minutes = std::stoi(sm[5]);
102             seconds = std::stoi(sm[6]);
103
104             end = ptm(date2, time_duration(hours, minutes, seconds));
105
106             boots += std::to_string(lines_scan) + "(" + file_name + "): ";
107             boots += end_date + "Boot Completed\n";
108
109             time_diff = end - begin;
110
111             boots += "\tBoot Time: ";
112             boots += std::to_string(time_diff.total_milliseconds()) + "ms \n\n";
113
114             boot_success++;
115
116             found_start = false;
117         }
118         lines_scan++;
119     }
120     file.close();
121 }
122 report += "Device Boot Report";
123 report += "\n\nIntouch log file: " + file_name + "\n";
124 report += "Lines scanned: " + std::to_string(lines_scan) + "\n\n";
125 report += "Device boot count: initiated = " + std::to_string(boot_total);
126 report += ", completed: " + std::to_string(boot_success) + "\n\n\n";
127
128 boots.erase(boots.end()-1);
129 report += boots;
130 std::ofstream out(output_name.c_str());
131 out << report;
132 out.close();
133
134 return 0;
135 }

```