

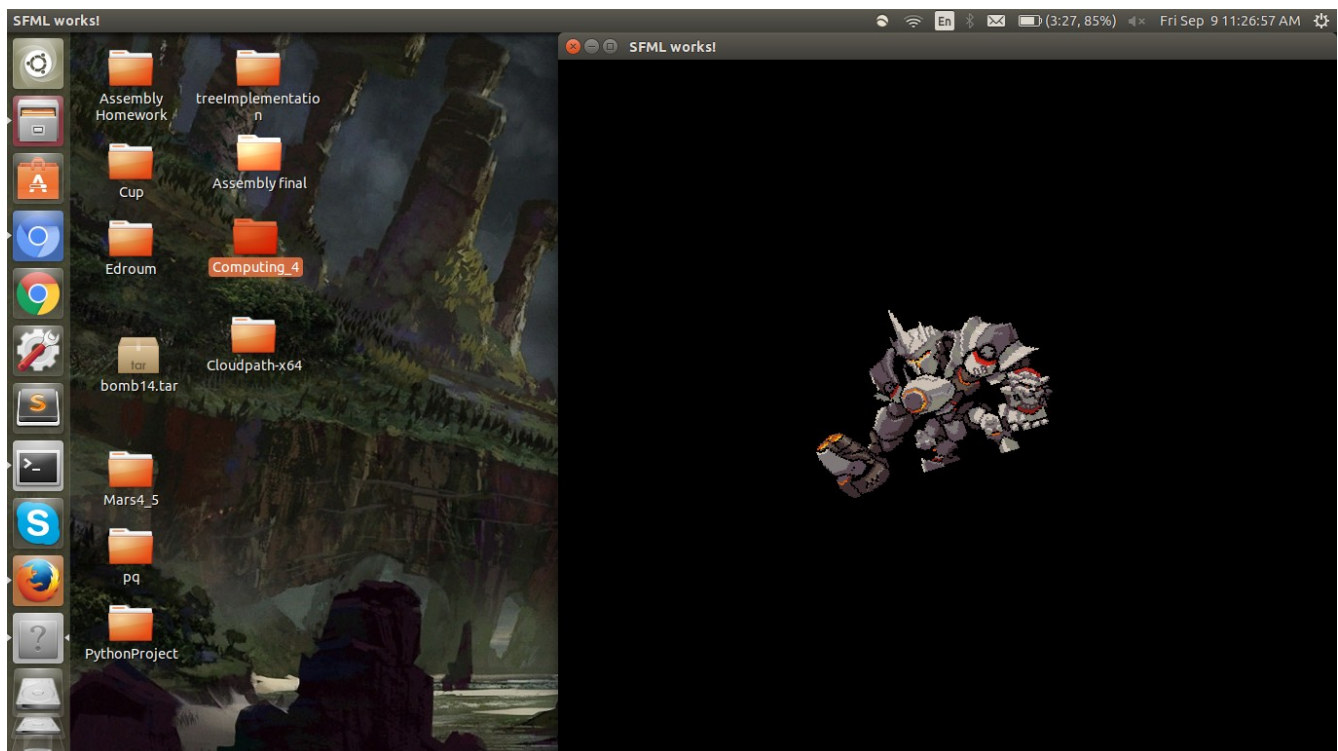
Contents:

1) PS0: <i>Hello World with SFML</i>	02
2) PS1: <i>Recursive Graphic</i>	05
3) PS2: <i>Linear Feedback Shift Register and Image Encoding</i>	15
4) PS3: <i>N-Body Simulation</i>	29
5) PS4: <i>Edit Distance</i>	45
6) PS5: <i>Ringer Buffer and Guitar Hero</i>	51
7) PS6: <i>Markov Model of Natural Language</i>	71
8) PS7: <i>Kronos Intouch Parsing</i>	79

PS0: Hello World with SFML

The assignment PS0 was to get familiar with our build environment and learn SFML. I accomplished this task by learning the documentation of SFML and was able to use keystrokes to move a picture in a window. We did not use any algorithms that were central to the assignment. It was not needed.

I learned in this assignment on what necessary libraries were needed to complete PS0. Understanding what each library offers and how I can use them in the future. For example, I learned how to use the `sf::Keyboard` function. Also understand more on how to insert pictures onto a window.



```
1: /*
2: Name: Albara Mehene
3: Date:9/8/2016
4: Computing IV
5: To move the sprite use arrow keys. To rotate it, use the A and S keys.
6: */
7:
8:
9:
10:
11: #include <SFML/Graphics.hpp>
12:
13: int main()
14: {
15:     sf::RenderWindow window(sf::VideoMode(800, 800), "SFML works!");
16:     /*sf::CircleShape shape(100.f);
17:     shape.setFillColor(sf::Color::Green);*/
18:     sf::Texture texture;
19:     //This sprite source is from a game called Overwatch. All credit goes to
Blizzard Entertainment & Overwatch.http://Overwatch.com
20:     if(!texture.loadFromFile("sprite.png"))
21:         return EXIT_FAILURE;
22:     sf::Sprite sprite(texture);
23:     int x ,y;
24:     x = 200;
25:     y = 200;
26:     sprite.setPosition(x,y);
27:
28:     //loop
29:     while (window.isOpen())
30:     {
31:         sf::Event event;
32:         while (window.pollEvent(event)){
33:
34:             if (event.type == sf::Event::Closed)
35:                 window.close();
36:         }
37:         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
38:             sprite.move(0,-3);
39:
40:         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
41:             sprite.move(0,3);
42:
43:         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
44:             sprite.move(-3,0);
45:
46:         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
47:             sprite.move(3,0);
48:
49:         if(sf::Keyboard::isKeyPressed(sf::Keyboard::A))
50:             sprite.rotate(-3);
51:
52:         if(sf::Keyboard::isKeyPressed(sf::Keyboard::S))
53:             sprite.rotate(3);
54:
55:         window.clear();
56:         window.draw(sprite);
57:         window.display();
58:
59:
60:
```

```
61:
62:     }
63:
64:     return 0;
65: }
```

PS1: Recursive Graphics (Sierpinski's Triangle)

In this assignment, we were to create a Sierpinski triangle by using recursion to draw the triangle. We needed be able to plot the base triangle and then recursively drawing smaller or bigger triangles outside or inside the base triangle. (Depending if you choose to make the base triangle small or large. We would then create our own original shape doing the same procedure as the Sierpinski.

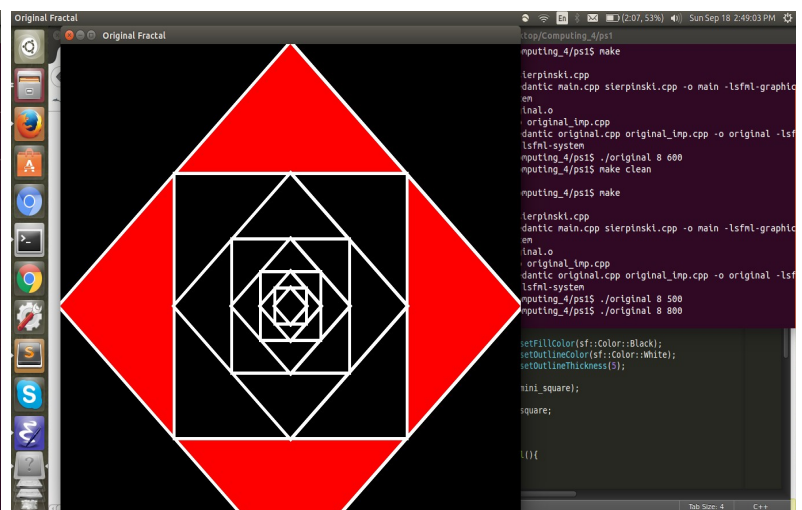
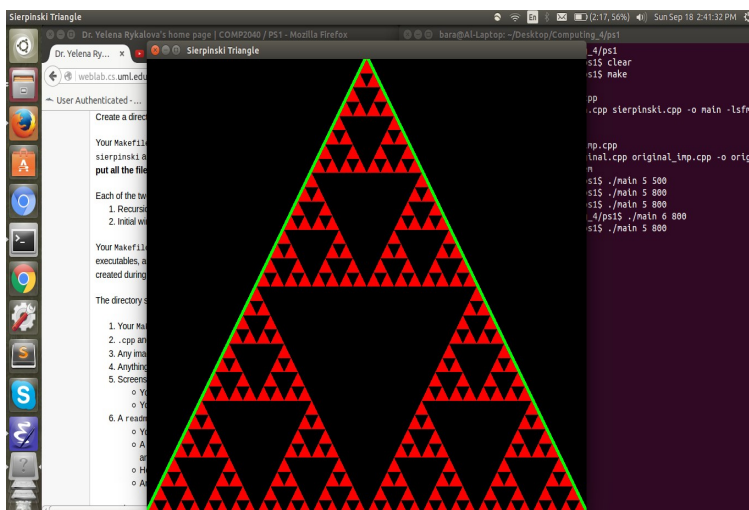
Sierpinski Triangle:

In the Sierpinski implantation, I created a class called Sierpinski that inherited the drawable library. This was used to be able to have a type convex shape to draw the base triangle and recursive triangle. The draw function was used to call each time we were to need to draw the function. When the user enter, I needed to set the point using the setPoint function in the sf::Convex Shape. In the recursive function, I would draw the points by dividing the 2 points to get the middle point. After drawing it, I would set the top, left, and right to different points to get the next mid points. Then take the amount of recursion depth and recursively call the function until 0.

Original:

I did everything the same idea as the Sierpinski triangle but I instead used 4 points instead of 3. I also changed the coordinates of the square to fit the screen.

In this assignment, I learned to figure out how to better use the sf::Convex Shape library, using functions like setPoint, getPoint, setFillColor, etc. I was able to better understand recursion and use it with other functions. Also learned the full use of type Vector2f.



```
1: cc = g++
2: #all
3:
4:
5: all : main original
6:
7: main : main.o sierpinski.o
8:      $(cc) -Wall -Werror -ansi -pedantic main.cpp sierpinski.cpp -o sierp
inski -lsfml-graphics -lsfml-window -lsfml-system
9:
10: main.o : sierpinski.hpp
11:      $(cc) -c main.cpp -o main.o
12:
13: sierpinski : sierpinski.cpp sierpinski.hpp
14:      $(cc) -c sierpinski.cpp -o sierpinski.o
15:
16:
17: original : original.o original_imp.o
18:      $(cc) -Wall -Werror -ansi -pedantic original.cpp original_imp.cpp -o
original -lsfml-graphics -lsfml-window -lsfml-system
19:
20: original.o : original.hpp
21:      $(cc) -c original.cpp -o original.o
22:
23: original_imp : original_imp.cpp original.hpp
24:      $(cc) -c original_imp.cpp -o original_imp.o
25:
26: clean:
27:      rm *.o main original
28:
29: run:
30:      ./sierpinski 5 800
31:      ./original 8 400
32:
33: debug: cc += -g
34: debug: main original
```

```
1:  /*
2:      Name: Albara Mehene
3:      Date: 9/17/2016
4:      Computing IV
5:  */
6:  #include <iostream>
7:  #include <SFML/Graphics.hpp>
8:  #include <SFML/Window.hpp>
9:  #include "sierpinski.hpp"
10:
11:  int main(int argc, char* argv[]){
12:
13:      if(argc < 3){
14:
15:          std::cout<<"sierpinski [recursion-depth] [side-length]
h]"<< std::endl;
16:          return -1;
17:      }
18:      //Atoi converts strings to integers
19:      int depth = atoi(argv[1]);
20:      int side = atoi(argv[2]);
21:
22:      sf::RenderWindow
23:          window(sf::VideoMode(side, (int)(.5*sqrt(3.)*(float)side)),
"Sierpinski Triangle");
24:
25:      Sierpinski sierpinski(depth, side);
26:
27:
28:      window.setFramerateLimit(1);
29:
30:      while(window.isOpen())
31:      {
32:          sf::Event event;
33:          while(window.pollEvent(event))
34:          {
35:              if(event.type == sf::Event::Closed)
36:                  window.close();
37:          }
38:          window.clear();
39:          window.draw(sierpinski);
40:          window.display();
41:      }
42:
43:      return 0;
44: }
```

```
1:
2:
3:
4:
5:
6:
7:
8:
9: #include <cmath>
10: #include <SFML/Graphics.hpp>
11: #include <SFML/Window.hpp>
12:
13: class Sierpinski : public sf::Drawable
14: {
15:     public:
16:
17:         Sierpinski(int N, int size_tri);
18:
19:         void sierpinski(sf::ConvexShape mid_triangle, int recursion,
sf::RenderTarget& target) const;
20:
21:         sf::ConvexShape filledtriangle(sf::Vector2f left_tri, sf::Ve
ctor2f bottom_tri, sf::Vector2f right_tri, sf::RenderTarget& target) const;
22:         //destructor;
23:         ~Sierpinski();
24:
25:     private:
26:         sf::ConvexShape triangle;
27:
28:         int depth_;
29:         int side_;
30:
31:
32:         void virtual draw(sf::RenderTarget& target, sf::RenderStates
states) const;
33:
34:
35: };
```



```
1: #include <iostream>
2: #include <cmath>
3: #include <SFML/Graphics.hpp>
4: #include <SFML/Window.hpp>
5: #include "sierpinski.hpp"
6:
7: void Sierpinski::draw(sf::RenderTarget& target, sf::RenderStates states) con
st{
8:     target.draw(triangle, states);
9:     sierpinski(triangle, depth_, target);
10:
11: }
12:
13: Sierpinski::Sierpinski(int N, int size_tri){
14:
15:     side_ = size_tri;
16:     depth_ = N;
17:
18:
19:     triangle.setPointCount(3);
20:     triangle.setPoint(0, sf::Vector2f(0,side_*(sqrt(3)/2)));//left
21:     triangle.setPoint(1, sf::Vector2f(side_,side_*(sqrt(3)/2)));//right
22:     triangle.setPoint(2, sf::Vector2f((side_/2),0));//top
23:
24:
25:     triangle.setFillColor(sf::Color::Red);
26:     triangle.setOutlineColor(sf::Color::Green);
27:     triangle.setOutlineThickness(5);
28: }
29:
30:
31: void Sierpinski::sierpinski(sf::ConvexShape mid_triangle, int recursion,sf::
RenderTarget& target) const{
32:     sf::Vector2f left,right,top;
33:     sf::Vector2f mid_lefttop, mid_leftright, mid_topright;
34:     sf::ConvexShape temp1_tri, temp2_tri, temp3_tri;
35:
36:     if(recursion == 0){
37:         return;
38:     }
39:     else{
40:         left = mid_triangle.getPoint(0);
41:         right = mid_triangle.getPoint(1);
42:         top = mid_triangle.getPoint(2);
43:
44:         mid_lefttop.x = (left.x + top.x)/2;
45:         mid_lefttop.y = (left.y + top.y)/2;
46:
47:         mid_leftright.x = (left.x + right.x)/2;
48:         mid_leftright.y = (left.y + right.y)/2;
49:
50:         mid_topright.x = (top.x + right.x)/2;
51:         mid_topright.y = (top.y + right.y)/2;
52:
53:         temp1_tri = filledtriangle(mid_lefttop,mid_leftright, mid_to
pright,target);
54:         temp2_tri = temp1_tri;
55:         temp3_tri = temp1_tri;
56:
57:         temp1_tri.setPoint(2,left);
58:         temp2_tri.setPoint(0,right);
```

```
59:         temp3_tri.setPoint(1,top);
60:
61:         sierpinski(temp1_tri, recursion - 1, target);
62:         sierpinski(temp2_tri, recursion - 1, target);
63:         sierpinski(temp3_tri, recursion - 1, target);
64:
65:
66:
67:
68:     }
69:
70:
71:
72: }
73:
74: sf::ConvexShape Sierpinski::filledtriangle(sf::Vector2f left_tri, sf::Vector
2f bottom_tri, sf::Vector2f right_tri,sf::RenderTarget& target) const{
75:
76:     sf::ConvexShape small_triangle;
77:
78:     small_triangle.setPointCount(3);
79:     small_triangle.setPoint(0, left_tri);
80:     small_triangle.setPoint(1, bottom_tri);
81:     small_triangle.setPoint(2, right_tri);
82:
83:     small_triangle.setFillColor(sf::Color::Black);
84:
85:     target.draw(small_triangle);
86:
87:     return small_triangle;
88: }
89:
90: Sierpinski::~Sierpinski(){
91:
92: }
```

```
1:  /*
2:      Name: Albara Mehene
3:      Date: 9/18/2016
4:      Computing IV
5:  */
6:
7:  #include <iostream>
8:  #include <cmath>
9:  #include <SFML/Graphics.hpp>
10: #include <SFML/Window.hpp>
11:
12: #include "original.hpp"
13:
14: int main(int argc, char* argv[]){
15:
16:     if(argc < 3){
17:
18:         std::cout<<"Fractal [recursion-depth] [side-length]"
<< std::endl;
19:         return -1;
20:     }
21:     //Atoi converts strings to integers
22:     int depth_ = atoi(argv[1]);
23:     int side_ = atoi(argv[2]);
24:
25:     sf::RenderWindow window(sf::VideoMode(side_,side_), "Original Fractal");
26:
27:     Fractal frac(depth_, side_);
28:
29:     window.setFramerateLimit(1);
30:
31:     while(window.isOpen())
32:     {
33:         sf::Event event;
34:         while(window.pollEvent(event))
35:         {
36:             if(event.type == sf::Event::Closed)
37:                 window.close();
38:         }
39:         window.clear();
40:         window.draw(frac);
41:         window.display();
42:     }
43:
44:     return 0;
45: }
```

```
1: #include <cmath>
2: #include <SFML/Graphics.hpp>
3: #include <SFML/Window.hpp>
4:
5: class Fractal : public sf::Drawable
6: {
7:     public:
8:
9:         Fractal(int n, int size_frac);
10:
11:         void fractal_rec(sf::ConvexShape fractal_shape, int recursion, sf::
RenderTarget &target) const;
12:
13:         sf::ConvexShape filledFractal(sf::Vector2f point1, sf::Vector2f poin
t2, sf::Vector2f point3, sf::Vector2f point4, sf::RenderTarget &target) const;
14:
15:         ~Fractal();
16:
17:
18:
19:
20:     private:
21:         sf::ConvexShape square;
22:
23:         int depth;
24:         int side;
25:
26:         void virtual draw(sf::RenderTarget& target, sf::RenderStates states)
const;
27:
28: };
```

```
1: #include <iostream>
2: #include <cmath>
3: #include <SFML/Graphics.hpp>
4: #include <SFML/Window.hpp>
5:
6: #include "original.hpp"
7:
8:
9: void Fractal::draw(sf::RenderTarget& target, sf::RenderStates states) const{
10:     target.draw(square, states);
11:     fractal_rec(square, depth, target);
12: }
13:
14: Fractal::Fractal(int N, int size_frac){
15:     side = size_frac;
16:     depth = N;
17:
18:     square.setPointCount(4);
19:     square.setPoint(0, sf::Vector2f(side/2, 0)); //top
20:     square.setPoint(1, sf::Vector2f(0, side/2)); //left
21:     square.setPoint(2, sf::Vector2f(side/2, side)); //bottom
22:     square.setPoint(3, sf::Vector2f(side, side/2)); //right
23:
24:     square.setFillColor(sf::Color::Red);
25:     square.setOutlineColor(sf::Color::White);
26:     square.setOutlineThickness(5);
27: }
28: void Fractal::fractal_rec(sf::ConvexShape fractal_shape, int recursion, sf::
RenderTarget &target) const{
29:     sf::Vector2f left,top,right,bottom;
30:     sf::Vector2f mid_lefttop, mid_topright, mid_rightbottom, mid_bottoml
eft;
31:
32:     sf::ConvexShape temp1_sqr, temp2_sqr, temp3_sqr, temp4_sqr;
33:
34:     if(recursion == 0){
35:         return;
36:     }
37:     else{
38:         top = fractal_shape.getPoint(0);
39:         left = fractal_shape.getPoint(1);
40:         bottom = fractal_shape.getPoint(2);
41:         right = fractal_shape.getPoint(3);
42:
43:         mid_lefttop.x = (left.x + top.x)/2;
44:         mid_lefttop.y = (left.y + top.y)/2;
45:
46:         mid_topright.x = (top.x + right.x)/2;
47:         mid_topright.y = (top.y + right.y)/2;
48:
49:         mid_rightbottom.x = (right.x + bottom.x)/2;
50:         mid_rightbottom.y = (right.y + bottom.y)/2;
51:
52:         mid_bottomleft.x = (left.x + bottom.x)/2;
53:         mid_bottomleft.y = (left.y + bottom.y)/2;
54:
55:         temp1_sqr = filledFractal(mid_lefttop, mid_topright, mid_rig
htbottom, mid_bottomleft, target);
56:
57:         fractal_rec(temp1_sqr, recursion - 1, target);
58:     }
```

```
59:
60: }
61: sf::ConvexShape Fractal::filledFractal(sf::Vector2f point1,sf::Vector2f poin
t2, sf::Vector2f point3, sf::Vector2f point4,sf::RenderTarget &target) const{
62:     sf::ConvexShape mini_square;
63:
64:     mini_square.setPointCount(4);
65:     mini_square.setPoint(0, point1);
66:     mini_square.setPoint(1, point2);
67:     mini_square.setPoint(2, point3);
68:     mini_square.setPoint(3, point4);
69:
70:     mini_square.setFillColor(sf::Color::Black);
71:     mini_square.setOutlineColor(sf::Color::White);
72:     mini_square.setOutlineThickness(5);
73:
74:     target.draw(mini_square);
75:
76:     return mini_square;
77:
78: }
79:
80: Fractal::~Fractal(){
81:
82: }
```

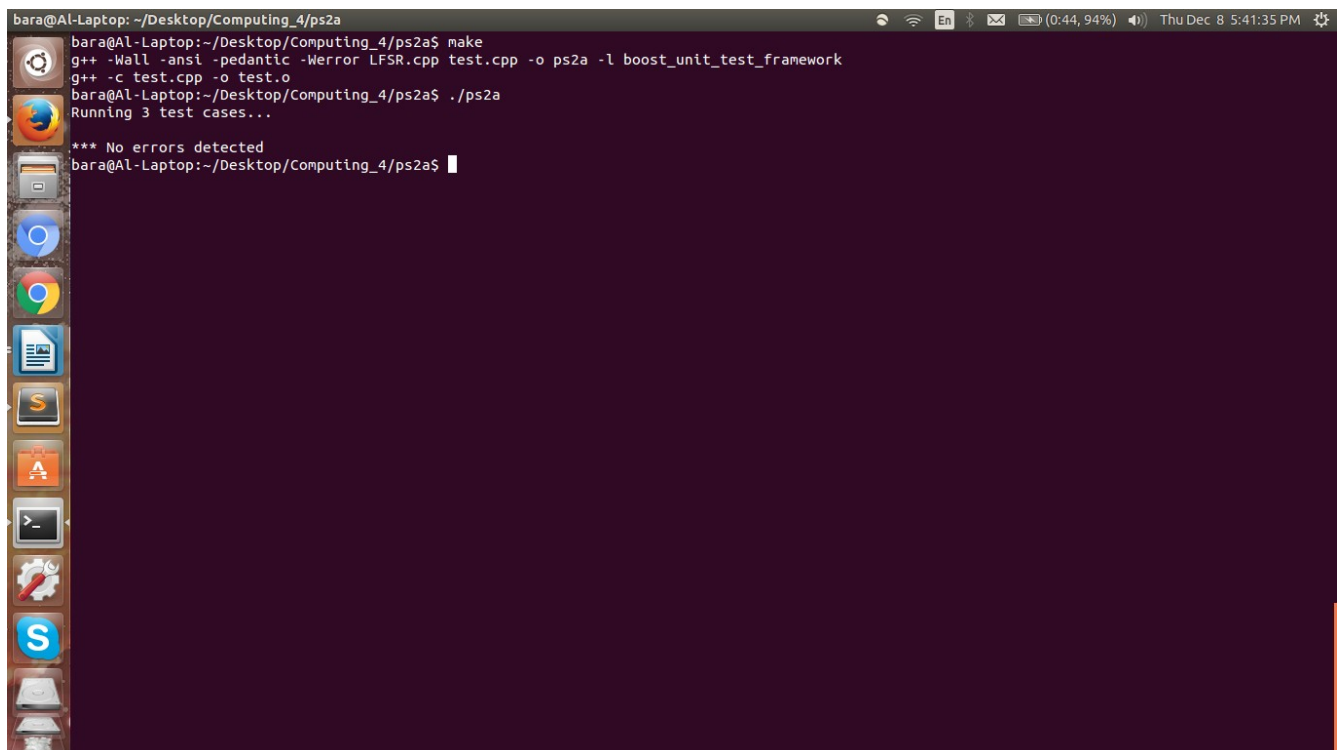
PS2: Linear Feedback Shift Register and Image Encoding

PS2a:

This assignment was to learn to register bits that shifts bits one position to the left 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. To test our LFSR is working, we needed to use software called boost.

I used this a class that accepts the seed and number of taps for the constructor. The step function would LFSR only once and would return the bit to test in my boost file. The generate function was essentially call the step function but would also give a condition.

I learned about what LFSR was. One big thing I learned is boost. I learned how create test cases having different conditions to test out the step and generate functions. Though, I currently can't find any use for it in my own projects.



```
bara@Al-Laptop: ~/Desktop/Computing_4/ps2a
bara@Al-Laptop:~/Desktop/Computing_4/ps2a$ make
g++ -Wall -ansi -pedantic -Werror LFSR.cpp test.cpp -o ps2a -l boost_unit_test_framework
g++ -c test.cpp -o test.o
bara@Al-Laptop:~/Desktop/Computing_4/ps2a$ ./ps2a
Running 3 test cases...
*** No errors detected
bara@Al-Laptop:~/Desktop/Computing_4/ps2a$
```

The image shows a terminal window on a Linux system. The user is in the directory ~/Desktop/Computing_4/ps2a. They run 'make' to compile the program, which uses g++ with various flags and links against the boost unit test framework. Then they run './ps2a' which executes the program and reports 'Running 3 test cases...' followed by '*** No errors detected'. The terminal has a dark purple background and a sidebar with application icons on the left. The top status bar shows system information like battery level and time.

```
1: cc = g++
2: #all
3:
4: all : LFSR test
5:
6: LFSR : LFSR.o test.o
7:      $(cc) -Wall -ansi -pedantic -Werror LFSR.cpp test.cpp -o ps2a -l boo
st_unit_test_framework
8:
9: LFSR.o : LFSR.hpp
10:      $(cc) -c LFSR.cpp -o LFSR.o
11:
12: test : test.cpp LFSR.hpp
13:      $(cc) -c test.cpp -o test.o
14:
15:
16: clean:
17:      rm *.o ps2a
18:
19: run:
20:      ./ps2a
21:
22: debug: cc += -g
23: debug: ps2a
24:
```



```
1:
2: #include "LFSR.hpp"
3:
4: #define BOOST_TEST_DYN_LINK
5: #define BOOST_TEST_MODULE Main
6: #include <boost/test/unit_test.hpp>
7:
8: BOOST_AUTO_TEST_CASE(fiveBitsTapAtTwo) {
9:     //steps through the function 8 times and checks to see the return bit is 1
or 0.
10:    LFSR l("00111", 2);
11:    BOOST_REQUIRE(l.step() == 1);
12:    BOOST_REQUIRE(l.step() == 1);
13:    BOOST_REQUIRE(l.step() == 0);
14:    BOOST_REQUIRE(l.step() == 0);
15:    BOOST_REQUIRE(l.step() == 0);
16:    BOOST_REQUIRE(l.step() == 1);
17:    BOOST_REQUIRE(l.step() == 1);
18:    BOOST_REQUIRE(l.step() == 0);
19:
20:    //Doing the math, it turns out to 198
21:    LFSR l2("00111", 2);
22:    BOOST_REQUIRE(l2.generate(8) == 198);
23:
24: }
25:
26: BOOST_AUTO_TEST_CASE(Test1){
27:     // I stepped 5 times by doing the xor to the 2nd tap.
28:    LFSR l3("0110110", 2);
29:    BOOST_REQUIRE(l3.step() == 1);
30:    BOOST_REQUIRE(l3.step() == 0);
31:    BOOST_REQUIRE(l3.step() == 1);
32:    BOOST_REQUIRE(l3.step() == 1);
33:    BOOST_REQUIRE(l3.step() == 1);
34:
35: }
36: BOOST_AUTO_TEST_CASE(Test2){
37:     //goes through step 3 times and by doing it out by hand, the answer is 7
38:     //It steps 3 times.
39:    LFSR l4("0110001", 3);
40:    BOOST_REQUIRE(l4.generate(4) == 7);
41: }
42:
43:
```

```
1:
2:
3: #include <iostream>
4: #include <string>
5: #include <cmath>
6:
7:
8: class LFSR {
9:
10: public:
11:     LFSR(std::string seed_, int tap_);
12:     ~LFSR();
13:     int step();
14:     int generate(int k);
15:
16:     friend std::ostream& operator<< (std::ostream &out, LFSR &lfsr);
17:
18: private:
19:     std::string seed;
20:     int tap;
21: };
22:
```

```
1: /*
2: Name: ALbara Mehene
3: Date: 9/25/2016
4: Computing IV
5:
6: */
7:
8:
9: #include <iostream>
10: #include <string>
11: #include <cmath>
12: #include "LFSR.hpp"
13:
14: //constructor
15: LFSR::LFSR(std::string seed_, int tap_){
16:
17:     seed = seed_;
18:     tap = tap_;
19:
20: }
21:
22:
23: int LFSR::step(){
24:     int bit;
25:     int size;
26:
27:     size = seed.length();//stored the amount of elements
28:
29:     bit = seed.at(0) ^ seed[size - tap - 1]; //Took the total elements
and subtracted by the tap and by 1
30:
31:
32:     seed.erase(0, 1); // erased the front element
33:
34:     if(bit == 1){//condition if its 1, it would return the chracter 1
35:         seed.push_back('1');
36:     }
37:     else{//returns 0 if its anything else
38:         seed.push_back('0');
39:     }
40:     //returns bit to test the test.cpp
41:     return bit;
42: }
43:
44: int LFSR::generate(int k){
45:     int temp = 0;
46:
47:     //Condition to test the generate function in test.cpp
48:     for(int i = k - 1; i >= 0; i--){
49:         if(step() == 1){
50:             temp += pow(2,i);
51:         }
52:     }
53:     return temp;
54:
55: }
56: //prints out the string if I were to use the a main. It was not required in
this assigment
57: std::ostream& operator<< (std::ostream &out, LFSR &lfsr){
58:     out << lfsr.seed;
59:     return out;
```

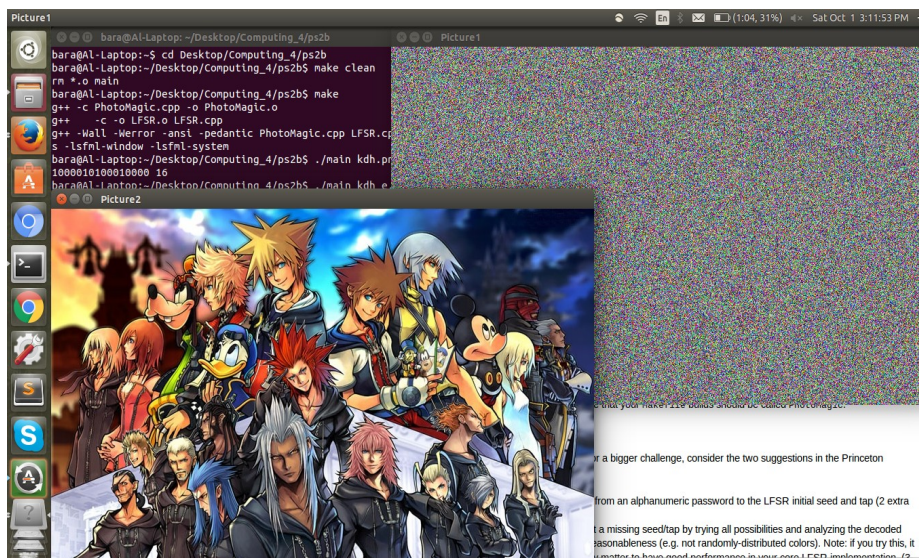
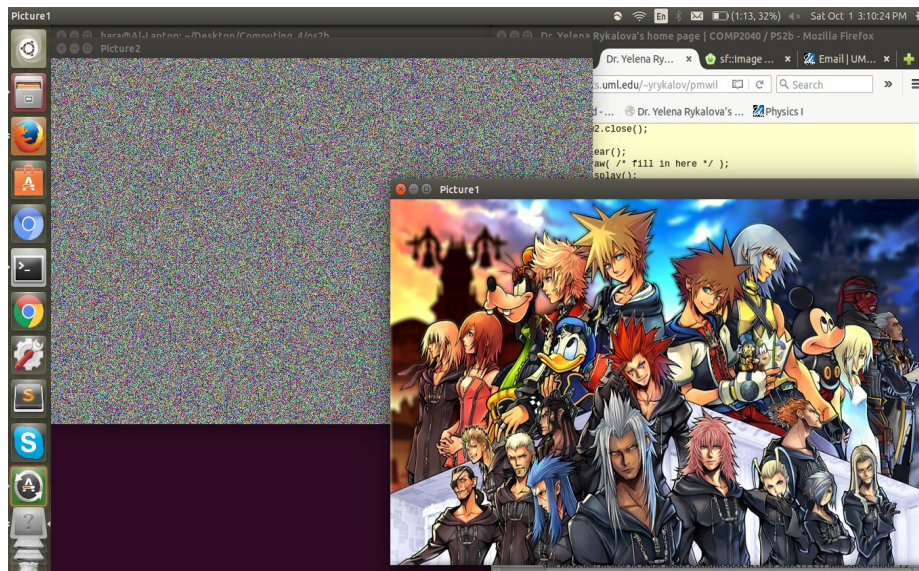
```
60: }  
61:  
62:  
63:  
64: LFSR::~LFSR() {  
65:  
66: }  
67:  
68:  
69:  
70:
```

PS2b:

Ps2b assignment was make a main client that uses LFSR to encrypt and decrypt pictures. To accomplish this task, we needed to pass a image. Each pixel would be passed go through LFSR.

To accomplish this task, I created a main and created a object of class LFSR. The accepted input of the image name would then be stored. I then used a function that would call the generate function and XOR the green, blue, and red. This would be done using the SFML Color type. Then create the output file and insert the new encrypted picture.

Learning about the amount of pixels in different types pictures was helpful to know in the future. XOR was nice to know what type of algorithm it is. Loading images and creating new image files was another skill I learned from this project.



Makefile **Sat Oct 01 15:26:16 2016** **1**

```
1: cc = g++
2: #all
3:
4: all : PhotoMagic
5:
6: PhotoMagic : PhotoMagic.o LFSR.o
7:             $(cc) -Wall -Werror -ansi -pedantic PhotoMagic.cpp LFSR.cpp -o Photo
Magic -lsfml-graphics -lsfml-window -lsfml-system
8:
9: PhotoMagic.o : LFSR.hpp
10:             $(cc) -c PhotoMagic.cpp -o PhotoMagic.o
11:
12: LFSR : LFSR.cpp LFSR.hpp
13:             $(cc) -c LFSR.cpp -o LFSR.o
14:
15: clean:
16:             rm *.o main
17:
```

```
1: /*
2: Name: Albara Mehene
3: Date: 10/1/2016
4: Computing IV
5:
6: */
7:
8: #include <SFML/System.hpp>
9: #include <SFML/Window.hpp>
10: #include <SFML/Graphics.hpp>
11:
12: #include "LFSR.hpp"
13:
14: sf::Image transform(sf::Image picture, LFSR lfsr);
15:
16: int main(int argc, char* argv[])
17: {
18:     if(argc < 5){
19:
20:         std::cout << "input-file.png, output-file.png, seed,
tap" << std::endl;
21:         return -1;
22:     }
23:
24:     std::string input = argv[1];
25:     std::string output = argv[2];
26:     std::string i_seed = argv[3];
27:     int i_tap = atoi(argv[4]);
28:
29:     sf::Image image;
30:     if (!image.loadFromFile(input))
31:         return -1;
32:
33:     sf::Image image_e = image;
34:     sf::Image temp_e;
35:
36:     //Pass the seed and tap function
37:     LFSR l(i_seed, i_tap);
38:     temp_e = transform(image_e, l);
39:
40:
41:     sf::Vector2u size = image.getSize();
42:     sf::Vector2u size2 = temp_e.getSize();
43:     sf::RenderWindow window(sf::VideoMode(size.x, size.y), "Picture1");
44:     sf::RenderWindow windowl(sf::VideoMode(size2.x, size2.y), "Picture2"
);
45:
46:
47:
48:
49:     sf::Texture texture;
50:     texture.loadFromImage(image);
51:
52:     sf::Texture texture_e;
53:     texture_e.loadFromImage(temp_e);
54:
55:     sf::Sprite sprite;
56:     sprite.setTexture(texture);
57:
58:     sf::Sprite sprite_e;
59:     sprite_e.setTexture(texture_e);
```

```
60:
61:     while (window.isOpen() && window1.isOpen())
62:     {
63:         sf::Event event;
64:         while (window.pollEvent(event))
65:         {
66:             if (event.type == sf::Event::Closed)
67:                 window.close();
68:         }
69:         while (window1.pollEvent(event))
70:         {
71:             if (event.type == sf::Event::Closed)
72:                 window1.close();
73:         }
74:
75:         window.clear(sf::Color::White);
76:         window1.clear(sf::Color::White);
77:         window.draw(sprite);
78:         window1.draw(sprite_e);
79:         window.display();
80:         window1.display();
81:     }
82: }
83:
84: // fredm: saving a PNG segfaults for me, though it does properly
85: // write the file
86: if (!temp_e.saveToFile(output))
87:     return -1;
88:
89: return 0;
90: }
91:
92: sf::Image transform(sf::Image picture, LFSR lfsr){
93:     // p is a pixel
94:     sf::Color p;
95:     int temp;
96:     sf::Vector2u size = picture.getSize();
97:
98:     // create photographic negative image of upper-left 200 px square
99:     for (unsigned int x= 0; x < size.x; x++) {
100:         for (unsigned int y = 0; y < size.y; y++) {
101:             p = picture.getPixel(x, y);
102:
103:             temp = lfsr.generate(8);
104:             p.r = p.r ^ temp;
105:
106:             temp = lfsr.generate(8);
107:             p.g = p.g ^ temp;
108:
109:             temp = lfsr.generate(8);
110:             p.b = p.b ^ temp;
111:
112:             picture.setPixel(x, y, p);
113:         }
114:     }
115:     return picture;
116: }
117:
118:
119:
120:
```


121:

```
1:
2:
3: #include <iostream>
4: #include <string>
5: #include <cmath>
6:
7:
8: class LFSR {
9:
10: public:
11:     LFSR(std::string seed_, int tap_);
12:     ~LFSR();
13:     int step();
14:     int generate(int k);
15:
16:     friend std::ostream& operator<< (std::ostream &out, LFSR &lfsr);
17:
18: private:
19:     std::string seed;
20:     int tap;
21: };
22:
```

```
1: /*
2: Name: ALbara Mehene
3: Date: 9/25/2016
4: Computing IV
5:
6: */
7:
8:
9: #include <iostream>
10: #include <string>
11: #include <cmath>
12: #include "LFSR.hpp"
13:
14: //constructor
15: LFSR::LFSR(std::string seed_, int tap_){
16:
17:     seed = seed_;
18:     tap = tap_;
19:
20: }
21:
22:
23: int LFSR::step(){
24:     int bit;
25:     int size;
26:
27:     size = seed.length();//stored the amount of elements
28:
29:     bit = seed.at(0) ^ seed[size - tap - 1]; //Took the total elements
and subtracted by the tap and by 1
30:
31:
32:     seed.erase(0, 1); // erased the front element
33:
34:     if(bit == 1){//condition if its 1, it would return the chracter 1
35:         seed.push_back('1');
36:     }
37:     else{//returns 0 if its anything else
38:         seed.push_back('0');
39:     }
40:     //returns bit to test the test.cpp
41:     return bit;
42: }
43:
44: int LFSR::generate(int k){
45:     int temp = 0;
46:
47:     //Condition to test the generate function in test.cpp
48:     for(int i = k - 1; i >= 0; i--){
49:         if(step() == 1){
50:             temp += pow(2,i);
51:         }
52:     }
53:     return temp;
54:
55: }
56: //prints out the string if I were to use the a main. It was not required in
this assigment
57: std::ostream& operator<< (std::ostream &out, LFSR &lfsr){
58:     out << lfsr.seed;
59:     return out;
```

```
60: }  
61:  
62:  
63:  
64: LFSR::~LFSR() {  
65:  
66: }  
67:  
68:  
69:  
70:
```

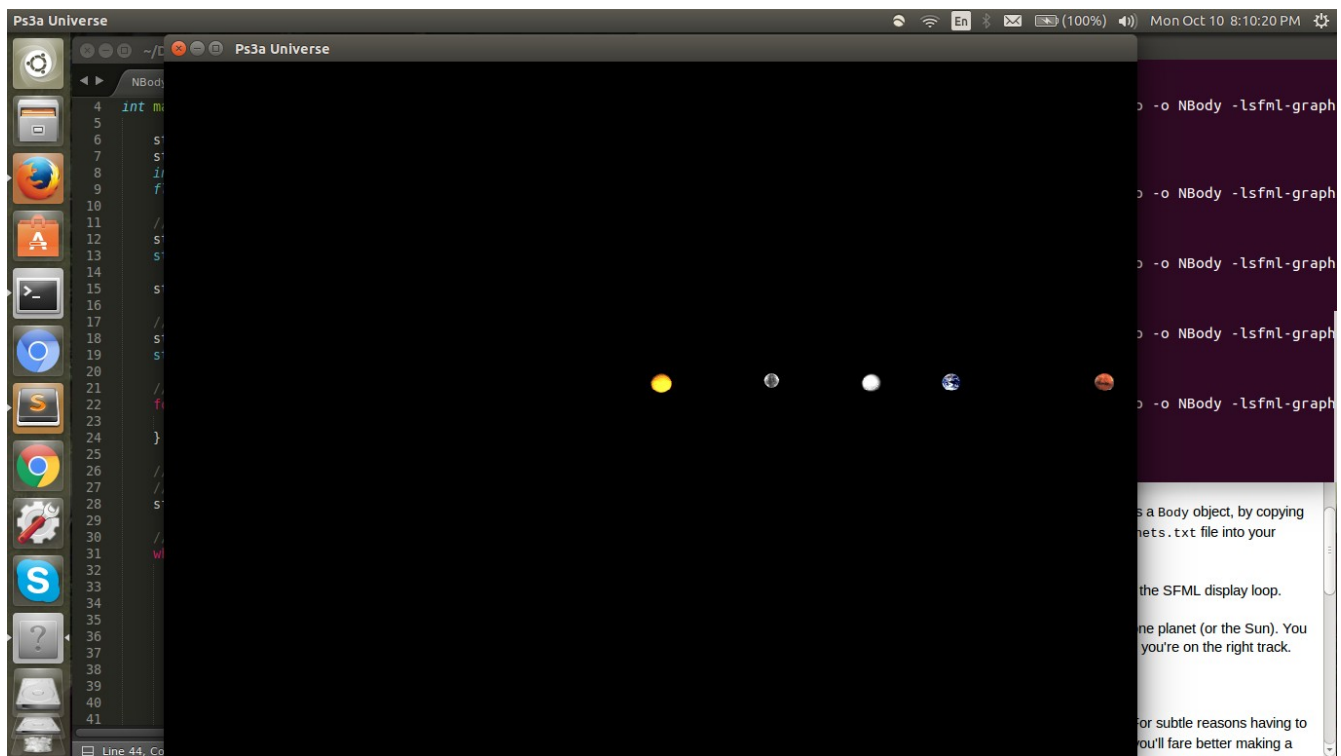
PS3: N-Body Simulation

Ps3a:

In the Ps3a, the assignment was to be able to make a static solar system using the coordinates from a text file. To accomplish this, we need to read each line and save each string into the associated variable. While doing that we would use the operator to be able to do that.

I used type vectors for the position and velocity because both had a x and y coordinate. I created a global variable that was the scale. We also needed to set each position by taking the screen size and setting each planet to the right position.

I learned that even with though the planets position numbers were huge, it could easily be placed by multiplying the position by scale and by the size of the window. I also learned that cin can actually read in a string and could read the next string just by using it again. This is done by giving the program a text file to read in.



Makefile **Fri Oct 07 14:58:11 2016** **1**

```
1: cc = g++
2:
3:
4: all : NBody_main
5:
6: NBody_main : NBody_main.o NBody.o
7:             $(cc) -Wall -Werror -ansi -pedantic NBody_main.cpp NBody.cpp -o NBod
y -lsfml-graphics -lsfml-window -lsfml-system
8:
9: NBody_main.o : NBody.hpp
10:             $(cc) -c NBody_main.cpp -o NBody_main.o
11:
12: NBody : NBody.cpp NBody.hpp
13:             $(cc) -c NBody.cpp -o NBody.o
14:
15: clean:
16:             rm *.o NBody
```

```
1: #include "NBody.hpp"
2:
3:
4: int main(int argc, char* argv[]){
5:
6:     std::string store;//stores the input from the file
7:     std::string name;//name of planet
8:     int numberOfPlanets;
9:     float radius;//radius of window
10:
11:     //stores the number of planets
12:     std::cin >> store;
13:     std::stringstream(store) >> numberOfPlanets;
14:
15:     std::vector<Body> objects(numberOfPlanets);//vector of objects to store
all objects
16:
17:     //stores the radius of the window
18:     std::cin >> store;
19:     std::stringstream(store) >> radius;
20:
21:     //loop that stores all relevant data from the file
22:     for (int x = 0; x < numberOfPlanets; x++){
23:         std::cin >> objects[x];
24:     }
25:
26:     //take the data inside the
27:     //vector of bodies and print it on the screen using SFML
28:     sf::RenderWindow window(sf::VideoMode(1000, 800), "Ps3a Universe");
29:
30:     //display window
31:     while(window.isOpen()){
32:         sf::Event event;
33:         while(window.pollEvent(event)){
34:             if(event.type == sf::Event::Closed)
35:                 window.close();
36:         }
37:         window.clear(sf::Color::Black);
38:         //display all the planets
39:         for(int i = 0; i < numberOfPlanets; i++){
40:             window.draw(objects[i]);
41:         }
42:         window.display();
43:     }
44:
45:     return 0;
46: }
```

```
1: #ifndef NBODY_H
2: #define NBODY_H
3:
4:
5: #include <SFML/Graphics.hpp>
6: #include <SFML/Window.hpp>
7: #include <iostream>
8: #include <string>
9: #include <vector>
10: #include <sstream>
11:
12:
13: class Body : public sf::Drawable
14: {
15:     private:
16:         double _time;
17:         sf::Vector2f _position;
18:         sf::Vector2f _velocity;
19:         float _mass;
20:         std::string _filename;
21:
22:     public:
23:         Body(float xCoord, float yCoord, float xVelocity, float yVelocity, float m
ass, std::string fileName);
24:
25:         Body();
26:         //friend istream &operator>>( istream &input, const Body &B);
27:         virtual void draw(sf::RenderTarget &target, sf::RenderStates states) const
;
28:
29:         //input stream overloader
30:         friend std::istream& operator>>(std::istream& in, Body& Body);
31:
32:         void setTime(double sTime);
33:         double getTime();
34:
35:         void setPosition(sf::Vector2f sPosition);
36:         sf::Vector2f getPosition();
37:
38:         void setVel(sf::Vector2f sVel);
39:         sf::Vector2f getVel();
40:
41:         void setMass(float sMass);
42:         float getMass();
43:
44:         void setFilename(std::string sFile);
45:         std::string getFilename();
46:
47:         ~Body();
48: };
49:
50: #endif
```



```
1: #include "NBody.hpp"
2: //.0000000002
3: const float SCALE = (2.50e+11); //This number negates the e+10 in the x posit
ion.
4:
5: void Body::draw(sf::RenderTarget &target, sf::RenderStates states) const{
6:     sf::Image image;
7:     sf::Texture texture;
8:     sf::Sprite sprite;
9:
10:    //std::cout << _position.x << _position.y << _filename << std::endl;
11:
12:    if(!image.loadFromFile(_filename)){
13:        std::cout << "ERROR: could not load image from file" << std::endl;
14:        return;
15:    }
16:
17:    texture.loadFromFile(_filename);
18:    sprite.setTexture(texture);
19:
20:    //need to multiply the x position by SCALE so the planets are not off th
e screen
21:    sprite.setPosition((_position.x/SCALE) * 500 + target.getSize().x/2, (_po
sition.y/SCALE) * 400 + target.getSize().y/2);
22: //x position / universe size * window size
23:    target.draw(sprite);
24:
25: }
26:
27: std::istream& operator>>(std::istream& in, Body& body){
28:
29:     in >> body._position.x >> body._position.y >> body._velocity.x >> body._v
elocity.y >> body._mass >> body._filename;
30:
31:     return in;
32:
33: }
34:
35: Body::Body(float xCoord, float yCoord, float xVelocity, float yVelocity, flo
at mass, std::string fileName){
36:     //setting vars to specifications
37:     _position.x = xCoord;
38:     _position.y = yCoord;
39:     _velocity.x = xVelocity;
40:     _velocity.y = yVelocity;
41:     _mass = mass;
42:     _filename = fileName;
43:
44: }
45: Body::Body(){
46:
47: }
48:
49: sf::Vector2f Body::getPosition(){
50:     return _position;
51: }
52:
53: sf::Vector2f Body::getVel(){
54:     return _velocity;
55: }
56:
```

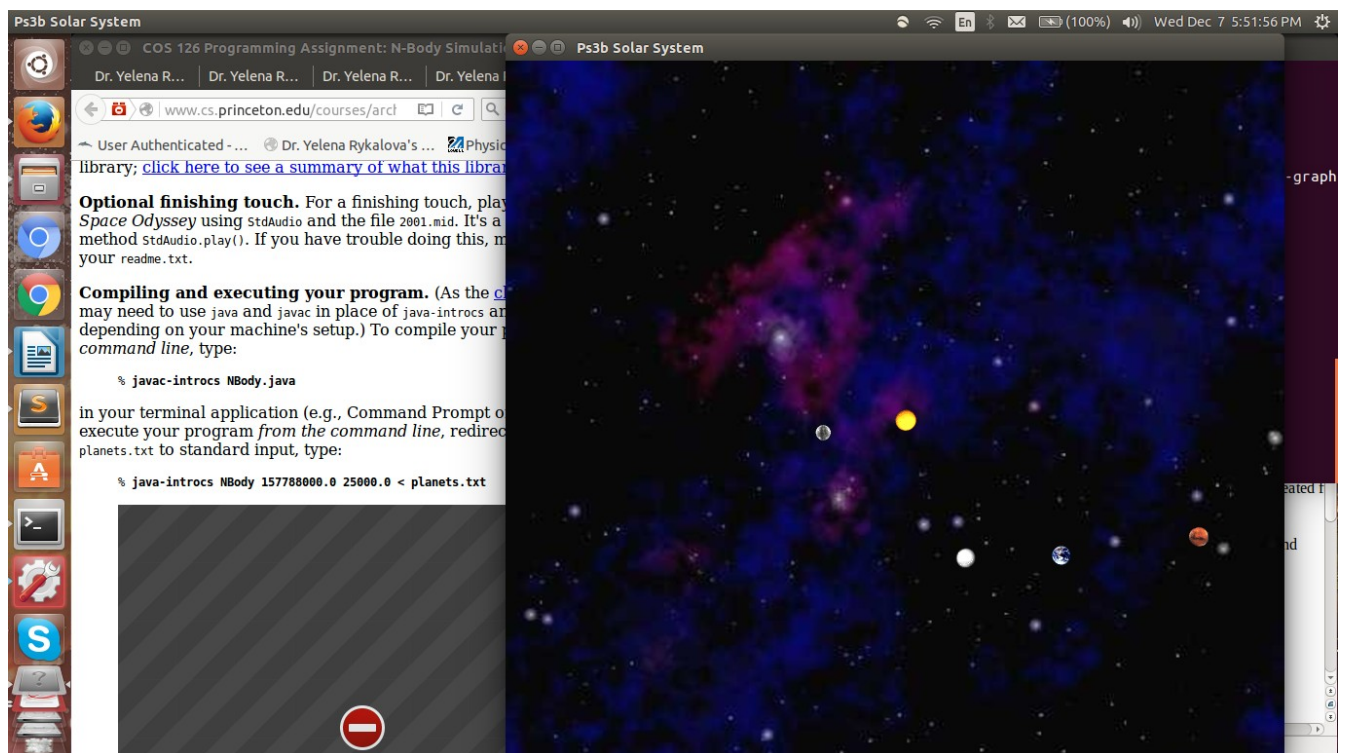
```
57: float Body::getMass(){
58:     return _mass;
59: }
60:
61: std::string Body::getFilename(){
62:     return _filename;
63: }
64:
65: Body::~~Body(){
66:
67: }
68:
69: //in main, before we make a new Body, we read in the file name that has the
proper characteristics, save them into multiple vars or strings, then feed them int
o the Body constructor
```

Ps3b:

In the Ps3b, this assignment was to be able to move the planets by giving the planets an acceleration and velocity. To accomplish this, we needed to use the Pairwise force, net force, and acceleration equations.

I used the equations to figure out what the acceleration was from each planet from the sun. Giving each planet its assigned velocity, position, mass, and the name of the file. After storing each planet's assigned values, I stored each planet in a vector of the number of planets. I created functions for the force, acceleration, and radius.

I never took physics ever in my high school. So learning about the right equations to figure out the net force and the acceleration. Learning to play sound once the window was open was useful and I will use in my later projects. Also, I learned that a vector can take any type. For example, it's possible to allocate a vector of objects.



```
1: cc = g++
2:
3:
4: all : NBody_main
5:
6: NBody_main : NBody_main.o NBody.o
7:             $(cc) -Wall -Werror -ansi -pedantic NBody_main.cpp NBody.cpp -o NBod
y -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
8:
9: NBody_main.o : NBody.hpp
10:             $(cc) -c NBody_main.cpp -o NBody_main.o
11:
12: NBody : NBody.cpp NBody.hpp
13:             $(cc) -c NBody.cpp -o NBody.o
14:
15: clean:
16:             rm *.o NBody
```

```
1: #include "NBody.hpp"
2:
3: double radius(sf::Vector2f pos1, sf::Vector2f pos2);
4: sf::Vector2f force(double mass1, double mass2, double radius, sf::Vector2f d
elta_p);
5: sf::Vector2f accel(double mass, sf::Vector2f p_force);
6: sf::Vector2f changeInPosition(sf::Vector2f pos1, sf::Vector2f pos2);
7:
8: const double G = (6.67e-11);
9:
10: int main(int argc, char* argv[]){
11:
12:
13:     if (argc < 3){
14:         std::cout << "max_time , time increment" << std::endl;
15:         return -1;
16:     }
17:
18:     double max_time = atof(argv[1]);
19:     double step_time = atof(argv[2]);
20:     double start_time = 0;
21:
22:     std::string store;//stores the input from the file
23:     std::string name;//name of planet
24:     int numberOfPlanets;
25:     double radius_of_window;//radius of window
26:     sf::Vector2f tempForce;
27:     double c_radius;
28:     sf::Vector2f c_force;
29:     sf::Vector2f c_accel;
30:     sf::Vector2f delta_p;
31:
32:     //stores the number of planets
33:     std::cin >> store;
34:     std::stringstream(store) >> numberOfPlanets;
35:
36:     std::vector<Body> objects(numberOfPlanets);//vector of objects to store
all objects
37:
38:     //stores the radius of the window
39:     std::cin >> store;
40:     std::stringstream(store) >> radius_of_window;
41:
42:     //loop that stores all relevant data from the file
43:     for (int x = 0; x < numberOfPlanets; x++){
44:         std::cin >> objects[x];
45:     }
46:
47:     //take the data inside the
48:     //vector of bodies and print it on the screen using SFML
49:     sf::RenderWindow window(sf::VideoMode(800, 800), "Ps3b Solar System");
50:
51:     sf::Image background;
52:     if(!background.loadFromFile("starfield.jpg"))
53:         return -1;
54:     sf::Texture backtex;
55:     backtex.loadFromImage(background);
56:
57:     sf::Sprite backsprite;
58:     backsprite.setTexture(backtex);
59:
```

```
60:     //plays audio
61:     sf::SoundBuffer buff;
62:     buff.loadFromFile("st.wav");
63:     sf::Sound sound;
64:     sound.setBuffer(buff);
65:     sound.play();
66:
67:     //backsprite.setScale(500,500);
68:
69:     //display window
70:     while(window.isOpen()){
71:         sf::Event event;
72:         while(window.pollEvent(event)){
73:             if(event.type == sf::Event::Closed)
74:                 window.close();
75:         }
76:         window.clear();
77:         window.draw(backsprite);
78:         //as long as we don't go past the max time
79:         if(start_time < max_time){
80:             //display all the planets
81:             for(int i = 0; i < numberOfPlanets; i++){
82:                 window.draw(objects[i]);
83:             }
84:             //for every planet
85:             for(int i = 0; i < numberOfPlanets; i++){
86:                 //don't do it for the current planet
87:                 if(i != 3){
88:                     delta_p = changeInPosition(objects[i].getPosition(), objects[3].getPosition());
89:                     c_radius = radius(objects[i].getPosition(), objects[3].getPosition());
90:                     c_force = force(objects[i].getMass(), objects[3].getMass(), c_radius, delta_p);
91:                     c_accel = accel(objects[i].getMass(), c_force);
92:                     objects[i].setAccel(c_accel);
93:                     objects[i].step(step_time);
94:                 }
95:             }
96:             //run step() to calculate the new positions, update start_time
97:             start_time += step_time;
98:
99:         }
100:         window.display();
101:     }
102:
103:     return 0;
104: }
105:
106: sf::Vector2f changeInPosition(sf::Vector2f pos1, sf::Vector2f pos2){
107:     sf::Vector2f change_p;
108:
109:     change_p.x = pos2.x - pos1.x;
110:     change_p.y = pos2.y - pos1.y;
111:
112:     return change_p;
113: }
114:
115: double radius(sf::Vector2f pos1, sf::Vector2f pos2){
116:     return std::sqrt(std::pow(pos1.x - pos2.x, 2) + std::pow(pos1.y - pos2.y, 2));
};
```

```
117: }
118:
119: sf::Vector2f force(double mass1, double mass2, double radius, sf::Vector2f de
lta_p){
120:     double F = (G * (mass1 * mass2)/(std::pow(radius,2)));
121:
122:     sf::Vector2f f_temp;
123:
124:     f_temp.x = F * (delta_p.x/radius);
125:     f_temp.y = F * (delta_p.y/radius);
126:
127:     return f_temp;
128:
129:
130: }
131: sf::Vector2f accel(double mass, sf::Vector2f p_force){
132:     sf::Vector2f cook;
133:
134:     cook.x = p_force.x/mass;
135:     cook.y = p_force.y/mass;
136:
137:     return cook;
138: }
```

```
1: #ifndef NBODY_H
2: #define NBODY_H
3:
4:
5: #include <SFML/Graphics.hpp>
6: #include <SFML/Window.hpp>
7: #include <SFML/Audio.hpp>
8: #include <iostream>
9: #include <string>
10: #include <vector>
11: #include <sstream>
12: #include <cmath>
13:
14: class Body : public sf::Drawable
15: {
16: private:
17:     double _time;
18:     sf::Vector2f _position;
19:     sf::Vector2f _velocity;
20:     sf::Vector2f _acceleration;
21:     double _mass;
22:     std::string _filename;
23:     sf::Vector2f _netforce;
24:     int _numberOfPlanets;
25:     sf::Vector2f _updatedAcceleration;
26:     sf::Vector2f _updatedPosition;
27:     sf::Vector2f _updatedVelocity;
28:
29: public:
30:     Body(double xCoord, double yCoord, double xVelocity, double yVelocity, double mass, std::string fileName);
31:
32:     Body();
33:     //friend istream &operator>>( istream &input, const Body &B);
34:     virtual void draw(sf::RenderTarget &target, sf::RenderStates states) const
35: ;
36:
37:     //input stream overloader
38:     friend std::istream& operator>>(std::istream& in, Body& Body);
39:
40:     void step(double t_time);
41:
42:     void setNetforce(sf::Vector2f sNetforce);
43:     sf::Vector2f getNetforce();
44:
45:     void setNumPlanets(int sNoplanets);
46:     int getNumPlanets();
47:
48:     void setupdatePos(sf::Vector2f sUpdatepos);
49:     sf::Vector2f getupdatePos();
50:
51:     /*void Body::setupdateVel(sf::Vector2f sUpdateVel);
52:     sf::Vector2f getupdateVel();*/
53:
54:     /*void setupdatedAccel(sf::Vector2f sUpdatedAccel);
55:     sf::Vector2f getupdatedAccel();*/
56:
57:     void setTime(double sTime);
58:     double getTime();
59:
60:     void setPosition(sf::Vector2f sPosition);
```



```
60:     sf::Vector2f getPosition();
61:
62:     void setVel(sf::Vector2f sVel);
63:     sf::Vector2f getVel();
64:
65:     void setAccel(sf::Vector2f sAccel);
66:     sf::Vector2f getAccel();
67:
68:     void setMass(double sMass);
69:     double getMass();
70:
71:     void setFilename(std::string sFile);
72:     std::string getFilename();
73:
74:     ~Body();
75: };
76:
77: #endif
```

```
1: #include "NBody.hpp"
2: //.0000000002
3: const double SCALE = (3.5e+11); //This number negates the e+10 in the x posit
ion.
4: const double G = (6.67e-11);
5: void Body::draw(sf::RenderTarget &target, sf::RenderStates states) const{
6:     sf::Image image;
7:     sf::Texture texture;
8:     sf::Sprite sprite;
9:
10:    //std::cout << _position.x << _position.y << _filename << std::endl;
11:
12:    if(!image.loadFromFile(_filename)){
13:        std::cout << "ERROR: could not load image from file" << std::endl;
14:        return;
15:    }
16:
17:    texture.loadFromFile(_filename);
18:    sprite.setTexture(texture);
19:
20:    //need to multiply the x position by SCALE so the planets are not off th
e screen
21:    sprite.setPosition((_position.x/SCALE) * 500 + target.getSize().x/2, (_po
sition.y/SCALE) * 500 + target.getSize().y/2);
22:    //x position / universe size * window size
23:    target.draw(sprite);
24:
25: }
26:
27: std::istream& operator>>(std::istream& in, Body& body){
28:
29:     in >> body._position.x >> body._position.y >> body._velocity.x >> body._v
elocity.y >> body._mass >> body._filename;
30:
31:     return in;
32:
33: }
34:
35: Body::Body(double xCoord, double yCoord, double xVelocity, double yVelocity,
double mass, std::string fileName){
36:     //setting vars to specifications
37:     _position.x = xCoord;
38:     _position.y = yCoord;
39:     _velocity.x = xVelocity;
40:     _velocity.y = yVelocity;
41:     _mass = mass;
42:     _filename = fileName;
43:
44: }
45: Body::Body(){
46:
47: }
48:
49: void Body::step(double t_time){
50:
51:
52:     sf::Vector2f distance;
53:     sf::Vector2f o_accel = getAccel();
54:     sf::Vector2f oldVel = getVel();
55:     sf::Vector2f newVel;
56:     sf::Vector2f endVel;
```

```
57:     sf::Vector2f oldPosition = getPosition();
58:     sf::Vector2f newPosition;
59:
60:     distance.x = t_time * oldVel.x;
61:     distance.y = t_time * oldVel.y;
62:
63:     newPosition.x = distance.x + oldPosition.x;
64:     newPosition.y = distance.y + oldPosition.y;
65:
66:     newVel.x = o_accel.x * t_time;
67:     newVel.y = o_accel.y * t_time;
68:
69:     endVel.x = newVel.x + oldVel.x;
70:     endVel.y = newVel.y + oldVel.y;
71:
72:
73:     setPosition(newPosition);
74:
75:     setVel(endVel);
76:
77:
78: }
79:
80:
81: void Body::setNetforce(sf::Vector2f sNetforce){
82:     _netforce = sNetforce;
83: }
84: sf::Vector2f Body::getNetforce(){
85:     return _netforce;
86: }
87:
88: void Body::setNumPlanets(int sNoplanets){
89:     _numberOfPlanets = sNoplanets;
90: }
91: int Body::getNumPlanets(){
92:     return _numberOfPlanets;
93: }
94:
95: void Body::setupdatePos(sf::Vector2f sUpdatepos){
96:     _updatedPosition = sUpdatepos;
97: }
98: sf::Vector2f Body::getupdatePos(){
99:     return _updatedPosition;
100: }
101:
102:
103:
104: void Body::setPosition(sf::Vector2f sPosition){
105:     _position = sPosition;
106: }
107:
108: sf::Vector2f Body::getPosition(){
109:     return _position;
110: }
111:
112: void Body::setVel(sf::Vector2f sVelocity){
113:     _velocity = sVelocity;
114: }
115:
116: sf::Vector2f Body::getVel(){
117:     return _velocity;
```

```
118: }
119:
120: void Body::setAccel(sf::Vector2f sAccel){
121:     _acceleration = sAccel;
122: }
123:
124: sf::Vector2f Body::getAccel(){
125:     return _acceleration;
126: }
127:
128: void Body::setMass(double sMass){
129:     _mass = sMass;
130: }
131:
132: double Body::getMass(){
133:     return _mass;
134: }
135:
136: void Body::setFilename(std::string sFile){
137:     _filename = sFile;
138: }
139:
140: std::string Body::getFilename(){
141:     return _filename;
142: }
143:
144: Body::~~Body(){
145:
146: }
147:
```

148: //in main, before we make a new Body, we read in the file name that has the proper characteristics, save them into multiple vars or strings, then feed them into the Body constructor

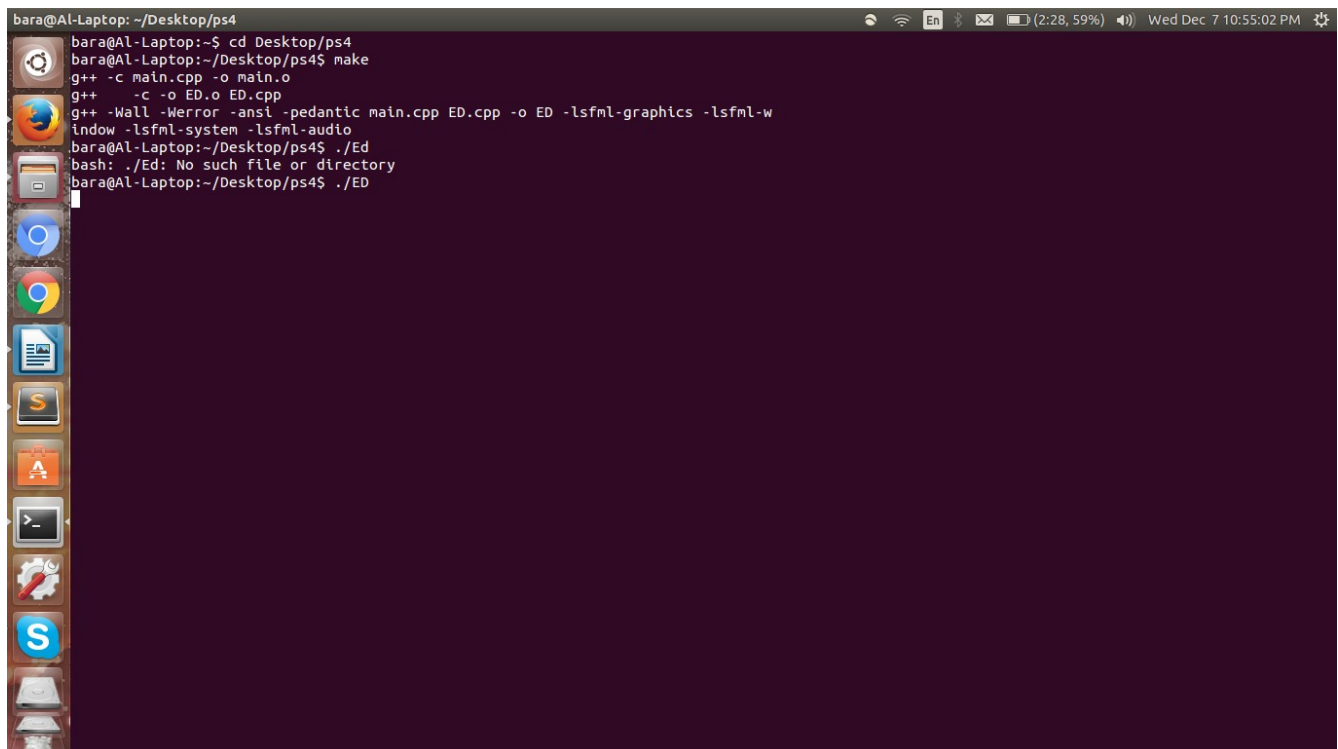
PS4: Edit Distance

In this assignment, this program is to create a optimal sequence alignment of two DNA strings. It will measure the similarity of two genetic sequences by the edit distance. It can be used for plagiarism detection, file revisioning. This is accomplished when two letters are aligned. If they are the same, it cost 0 bits. If they are different, it costs 1 bit. If their inset any at all, it will cost 2 bits.

This assignment could be approached in many different ways. Recursion and dynamic programming were recommend choices for this assignment. I tried taking the dynamic approach using an NxM matrix. Creating a class called ED, the constructor took two string were set to the member variables.

For the assignment, I didn't learn so much. However, I did learn the understand of edit distance. I learned what the necessary bits are needed, the alignment, and the total amount of distance needed for alignment.

Sadly, I was not able to finish this assignment. I believe I was confused on how to approach this assignment and felt lost on what to do next. When I tried running the program, I had a continues loop and had to close the the program by killing it.



```
bara@Al-Laptop: ~/Desktop/ps4
bara@Al-Laptop:~$ cd Desktop/ps4
bara@Al-Laptop:~/Desktop/ps4$ make
g++ -c main.cpp -o main.o
g++ -c -o ED.o ED.cpp
g++ -Wall -Werror -ansi -pedantic main.cpp ED.cpp -o ED -lsfml-graphics -lsfml-w
indow -lsfml-system -lsfml-audio
bara@Al-Laptop:~/Desktop/ps4$ ./Ed
bash: ./Ed: No such file or directory
bara@Al-Laptop:~/Desktop/ps4$ ./ED
```

```
1: cc = g++
2:
3: all : main
4:
5: main : main.o ED.o
6:      $(cc) -Wall -Werror -ansi -pedantic main.cpp ED.cpp -o ED -lsfml-gra
physics -lsfml-window -lsfml-system -lsfml-audio
7:
8: main.o : ED.hpp
9:      $(cc) -c main.cpp -o main.o
10:
11: ED : ED.cpp ED.hpp
12:      $(cc) -c ED.cpp -o ED.o
13:
14: clean:
15:      rm *.o ED
```

```
1:
2: #include "ED.hpp"
3:
4: int main(int argc, char* argv[]){
5:
6:     std::string store;
7:     std::string string_1;
8:     std::string string_2;
9:     std::string answer;
10:
11:     sf::Clock clock;
12:     sf::Time t;
13:
14:     std::cin >> store;
15:     string_1 = store;
16:     std::cin >> store;
17:     string_2 = store;
18:
19:     ED temp(string_1,string_2);
20:
21:     //displays the execution time
22:     t = clock.getElapsedTime();
23:     std::cout << t.asSeconds() << "seconds" << std::endl;
24:
25:
26:
27:     return 0;
28: }
```

```
1: #ifndef ED_H
2: #define ED_H
3:
4: #include <string>
5: #include <iostream>
6: #include <algorithm>
7: #include <SFML/System.hpp>
8:
9: class ED{
10: private:
11:     std::string _str1;
12:     int _str1Len;
13:     std::string _str2;
14:     int _str2Len;
15:
16:     int **_array;
17:
18: public:
19:     ED(std::string str1, std::string str2); //constructor, allocates any data structures
20:
21:     static int penalty(char a, char b); //returns penalty for aligning chars(0 or 1)
22:
23:     static int min(int a, int b, int c); //returns minimum of 3 args
24:
25:     int optDistance(); //populates matrices based on two strings, returns optimal distance
26:
27:     std::string alignment(); //traces the matrix and returns string to be printed
28:
29:     int getStr1Len();
30:
31:     int getStr2Len();
32:
33:     ~ED();
34:
35: };
36:
37: #endif
```



```
1: #include <string>
2: #include "ED.hpp"
3:
4: //constructor
5: ED::ED(std::string str1, std::string str2){
6:     //basic assignment
7:     _str1 = str1;
8:     _str2 = str2;
9:
10:    _str1Len = str1.size();
11:    _str2Len = str2.size();
12:
13:    //allocate for the first dimension
14:    _array = new int*[_str1Len + 1];///  
+1 because we assume the first column/row is empty
15:
16:    //need to allocate 2nd dimension of array, <= because _array is (str1Len +
17:
18: }
19:
20: //get the penalty for comparing the args
21: int ED::penalty(char a, char b){
22:
23:     //test = b - a;
24:     if(a == b) //check if chars are the same
25:         return 0;
26:     return 1;
27: }
28:
29: //returns minimum of the three args, basic comparison
30: int ED::min(int a, int b, int c){
31:     if(a < b && a < c)
32:         return a;
33:     if(b < c)
34:         return b;
35:     else
36:         return c;
37: }
38:
39: //traverses the 2d array, the meat of the program, have to move backwards (start at end)
40: /*int ED::optDistance(){
41:     //traverse from the end of the y-axis to the front
42:
43: }
44:
45: //traces the matrix and returns a string
46: std::string ED::alignment(){
47:
48:
49: }*/
50:
51: int ED::getStr1Len(){
52:     return _str1Len;
53: }
54:
55: int ED::getStr2Len(){
56:     return _str2Len;
57: }
58:
```

```
59: ED::~~ED(){
60:     //need to delete the memory we used for _array, this is second level
61:     //need to delete the first level
62:     delete[] _array;
63: }
```

PS5: Ringer Buffer and Guitar Hero

PS5a:

In this assignment, we needed implement the ring buffer that will hold the guitar string position data, and write test functions and exception handling. This assignment was to create the length of the string determining its fundamental frequency of vibration. To accomplish this task, I needed to create a ring buffer that will determine the fundamental frequency and the harmonics of the frequency.

The algorithms we used was a system called the cyclic wrap- around. This concept was to basically reuse an array without extending the amount of elements. I needed to have 2 pointers called the first and last arrays. Each time a user would enter a number in array, the pointer last would place the number in the array and go to the next one. When the user wants to takes out the first array, the first pointer would remove the first number and save the address of the next element. If the queue was full, I could set the pointer last to the beginning of the array. If it was full, it would notify the user its full and stop. Until the user dequeues the the value, the user can not enter any more numbers. If dequeue is continued to be called and the array is empty, the user will be notified that it is empty by throwing a exception.

The understanding of an array that wraps around to replace a element, it a great concept that could save memory in the future. It can be used to be efficient with memory. Whats even more great is that I figured out that members in a class can be accessed with needed gets and sets. Even though I should have known this before, figuring it out in this program helped out a lot. I also learned about throwing a exception when a error has occurred.

```
bara@AL-Laptop: ~/Desktop/ps5a
bara@AL-Laptop:~$ cd Desktop/ps51
bash: cd: Desktop/ps51: No such file or directory
bara@AL-Laptop:~$ cd Desktop/ps5a
bara@AL-Laptop:~/Desktop/ps5a$
bara@AL-Laptop:~/Desktop/ps5a$ make
g++ -Wall -ansi -pedantic -Werror RingBuffer.cpp test.cpp -o ps5a -l boost_unit_
test_framework
g++ -c test.cpp -o test.o
bara@AL-Laptop:~/Desktop/ps5a$ ls
cpplint.py  ps5a      RingBuffer.cpp  RingBuffer.o  test.o
Makefile    ps5a-readme.txt RingBuffer.hpp   test.cpp
bara@AL-Laptop:~/Desktop/ps5a$ ./ps5a
Running 3 test cases...
0
100
1
100
2
100
0
3
1
3
2
3
3
3
3
3
3
3
*** No errors detected
bara@AL-Laptop:~/Desktop/ps5a$
```

(Compiled with Makefile)

Makefile

Fri Oct 28 14:24:41 2016

1

```
1: cc = g++
2:
3: all : RingBuffer test
4:
5: RingBuffer : RingBuffer.o test.o
6:          $(cc) -Wall -ansi -pedantic -Werror RingBuffer.cpp test.cpp -o ps5a
-l boost_unit_test_framework
7:
8: RingBuffer.o : RingBuffer.hpp
9:          $(cc) -c RingBuffer.cpp -o RingBuffer.o
10:
11: test : test.cpp RingBuffer.hpp
12:       $(cc) -c test.cpp -o test.o
13:
14:
15: clean :
16:       rm *.o ps5a
```

```
1: /*Copyright [2016] <Albara Mehene> */
2: /*#define BOOST_TEST_DYN_LINK
3: #define BOOST_TEST_MODULE Main
4: */
5:
6: #define BOOST_TEST_DYN_LINK
7: #define BOOST_TEST_MODULE Main
8: #include <boost/test/unit_test.hpp>
9:
10: #include <stdint.h>
11: #include <iostream>
12: #include <string>
13: #include <exception>
14: #include <stdexcept>
15:
16: #include "RingBuffer.hpp"
17:
18: BOOST_AUTO_TEST_CASE(RBconstructor) {
19:     // normal constructor
20:     BOOST_REQUIRE_NO_THROW(RingBuffer(100));
21:
22:     // this should fail
23:     BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
24:     BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
25: }
26:
27: BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
28:     RingBuffer rb(100);
29:
30:     rb.enqueue(2);
31:     rb.enqueue(1);
32:     rb.enqueue(0);
33:
34:     BOOST_REQUIRE(rb.dequeue() == 2);
35:     BOOST_REQUIRE(rb.dequeue() == 1);
36:     BOOST_REQUIRE(rb.dequeue() == 0);
37:
38:     BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
39: }
40:
41: BOOST_AUTO_TEST_CASE(test1) {
42:     RingBuffer rb(3);
43:     rb.enqueue(2);
44:     rb.enqueue(3);
45:     rb.enqueue(4);
46:     BOOST_REQUIRE(rb.isFull());
47:     BOOST_REQUIRE_THROW(rb.enqueue(5), std::runtime_error)
48: }
```

```
1: /*Copyright [2016] <Albara Mehene> */
2:
3: #include "RingBuffer.hpp"
4:
5: // creates a empty ringbuffer with a max capacity
6: RingBuffer::RingBuffer(int capacity) {
7:     if(capacity < 1) {
8:         throw std::invalid_argument("capacity must be greater than zero");
9:     }
10:     cap = capacity;
11:     count = 0;
12:     array = new int16_t[capacity];
13:     first = array;
14:     last = array;
15: }
16:
17: // returns the number of items currently in the buffer
18: int RingBuffer::size() {
19:     return count;
20: }
21:
22: // checks to see if the buffer is empty
23: bool RingBuffer::isEmpty() {
24:     if (count == 0) {
25:         return 1;
26:     } else {
27:         return 0;
28:     }
29: }
30:
31: // checks to see if the buffer is full
32: bool RingBuffer::isFull() {
33:     if (count == cap) {
34:         return 1;
35:     } else {
36:         return 0;
37:     }
38: }
39:
40: // add item x to the end of buffer
41: void RingBuffer::enqueue(int16_t x) {
42:     if(isFull() == 1) {
43:         throw std::runtime_error("can't enqueue to a full ring");
44:     }
45:
46:     if (last == (array+(cap-1))) {
47:         count++;
48:         (*last) = x;
49:         last = array;
50:     } else {
51:         count++;
52:         (*last) = x;
53:         last = (last + 1);
54:     }
55: }
56:
57: // deletes and returns the item from the front of the buffer
58: int16_t RingBuffer::dequeue() {
59:     if (isEmpty() == 1) {
60:         throw std::runtime_error("can't dequeue to a empty ring");
61:     }
62: }
```

```
62:
63:     int16_t store;
64:
65:     if(first == (array+(cap-1))){
66:         count--;
67:         store = (*first);
68:         first = array;
69:         return store;
70:     }else{
71:         count--;
72:         store = (*first);
73:         first = (first+1);
74:         return store;
75:     }
76: }
77:
78: // returns item from the front without deleting it
79: int16_t RingBuffer::peek() {
80:     int16_t temp = (*first);
81:     return temp;
82: }
83:
84: RingBuffer::~RingBuffer() {
85:     delete[] array;
86: }
87: void RingBuffer::print_out(){
88:
89:     for(int i = 0; i < cap;i++){
90:         std::cout << "Array:  " << array[i] << std::endl;
91:     }
92: }
```



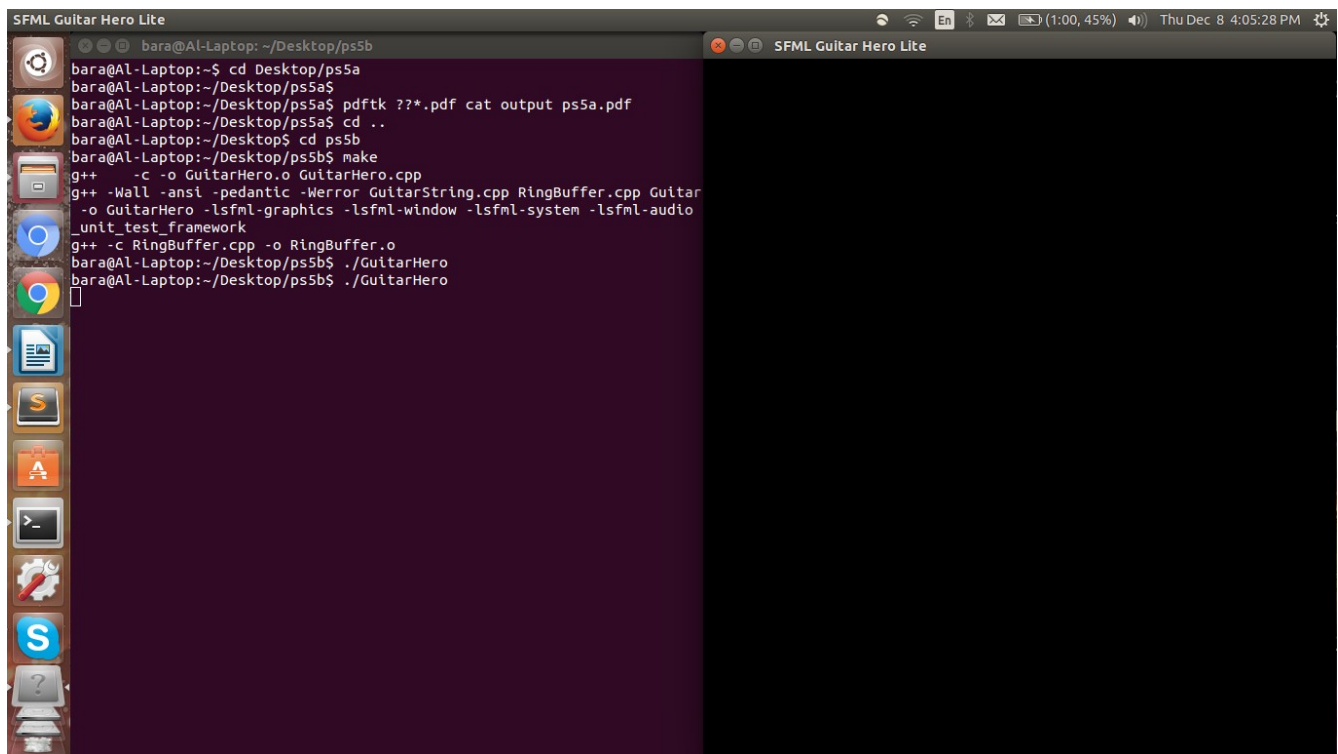
```
1: /*Copyright [2016] <Albara Mehene> */
2: #ifndef RINGBUFFER_HPP
3: #define RINGBUFFER_HPP
4:
5: #include <boost/test/unit_test.hpp>
6:
7: #include <SFML/Graphics.hpp>
8: #include <SFML/System.hpp>
9: #include <SFML/Audio.hpp>
10: #include <SFML/Window.hpp>
11: #include <stdint.h>
12: #include <iostream>
13: #include <string>
14: #include <exception>
15: #include <stdexcept>
16: #include <vector>
17:
18: class RingBuffer{
19: public:
20:     explicit RingBuffer(int capacity);
21:     int size();
22:     bool isEmpty();
23:     bool isFull();
24:     void enqueue(int16_t x);
25:     int16_t dequeue();
26:     int16_t peek();
27:     void print_out();
28:     ~RingBuffer();
29: private:
30:     int16_t *first;
31:     int16_t *last;
32:     int16_t *array;
33:     int count;
34:     int cap;
35: };
36:
37: #endif
```

PS5b:

The main purpose of this assignment is to actually make the simulation of the guitar string. This part we are plucking of the string and inserting white noise into every element. We needed to be able to use keys to play sound from each key. To accomplish this task, we needed to first create the guitar string of the given frequency using the sampling rate of 44,100Hz. Then another constructor that takes in a vector of type `int_16` (double). Which will initialize the contents of the buffer to the values. Then create a function that replaces the items in the buffer with random values between -32768 to 32767. Then a function to delete the sample at the front of the ring buffer and add to the end of the ring buffer the average of the first two samples, multiplied by the energy decay factor.

In this program, I created a pointer to a ring buffer to have sounds plucked in the buffer. I created a member variable to save the sampling rate divided by the given frequency. The first constructor, I enqueued 0 in each element of the amount the user wants for the frequency. In the other constructor, it would be called vectors of type `sf::Int16`. It would allocated a Ring Buffer of the size given. In the pluck function, I deleted the current Ring Buffer, and relocated a new Ring Buffer. Then I made a loop to insert white noise to each element. In the tic function, I created a 2 variables of type double that save the first value, and return the front of the array. I then added both and divided by 2 to multiply it by the decay. Then finally enqueueing the result value into the ring buffer. Then in the main, I had each key becomes registered and given a sound. To do this, I used the SFML library to use the keyboard function.

The Guitar String assignment helped me understand how to create sound and input them into any key. In the future, maybe creating my own program that can simulate different types of instruments. More understanding of the use of inheritance and the use of calling a function while allocating memory.



```
baraa@Al-Laptop:~$ cd Desktop/ps5a
baraa@Al-Laptop:~/Desktop/ps5a$
baraa@Al-Laptop:~/Desktop/ps5a$ pdftk ??*.pdf cat output ps5a.pdf
baraa@Al-Laptop:~/Desktop/ps5a$ cd ..
baraa@Al-Laptop:~/Desktop$ cd ps5b
baraa@Al-Laptop:~/Desktop/ps5b$ make
g++ -c -o GuitarHero.o GuitarHero.cpp
g++ -Wall -ansi -pedantic -Werror GuitarString.cpp RingBuffer.cpp Guitar
-o GuitarHero -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
_unit_test_framework
g++ -c RingBuffer.cpp -o RingBuffer.o
baraa@Al-Laptop:~/Desktop/ps5b$ ./GuitarHero
baraa@Al-Laptop:~/Desktop/ps5b$ ./GuitarHero
```

(Compiled with makefile)
(Extra credit for layout was not created)

```
1: cc = g++
2:
3: all : GuitarString RingBuffer GuitarHero
4:
5: GuitarString : GuitarString.o RingBuffer.o GuitarHero.o
6:      $(cc) -Wall -ansi -pedantic -Werror GuitarString.cpp RingBuffer.cpp
GuitarHero.cpp -o GuitarHero -lsfml-graphics -lsfml-window -lsfml-system -lsfml-aud
io -l boost_unit_test_framework
7:
8: GuitarString.o : GuitarString.hpp RingBuffer.hpp
9:      $(cc) -c GuitarString.cpp -o GuitarString.o
10:
11: RingBuffer : RingBuffer.hpp
12:      $(cc) -c RingBuffer.cpp -o RingBuffer.o
13:
14: GuitarHero : GuitarHero.cpp RingBuffer.hpp GuitarString.hpp
15:      $(cc) -c GuitarHero.cpp -o GuitarHero.o
16:
17:
18: clean :
19:      rm *.o GuitarHero debug
20:
21: debug :
22:      g++ -g -Wall -ansi -pedantic -Werror GuitarString.cpp RingBuffer.cpp
GStest.cpp -o debug -l boost_unit_test_framework
23:
```

```
1: /*Copyright [2016] <Albara Mehene> */
2: /*#define BOOST_TEST_DYN_LINK
3: #define BOOST_TEST_MODULE Main
4: */
5:
6: #define BOOST_TEST_DYN_LINK
7: #define BOOST_TEST_MODULE Main
8: #include <boost/test/unit_test.hpp>
9:
10: #include <stdint.h>
11: #include <iostream>
12: #include <string>
13: #include <exception>
14: #include <stdexcept>
15:
16: #include "RingBuffer.hpp"
17:
18: BOOST_AUTO_TEST_CASE(RBconstructor) {
19:     // normal constructor
20:     BOOST_REQUIRE_NO_THROW(RingBuffer(100));
21:
22:     // this should fail
23:     BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
24:     BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
25: }
26:
27: BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
28:     RingBuffer rb(100);
29:
30:     rb.enqueue(2);
31:     rb.enqueue(1);
32:     rb.enqueue(0);
33:
34:     BOOST_REQUIRE(rb.dequeue() == 2);
35:     BOOST_REQUIRE(rb.dequeue() == 1);
36:     BOOST_REQUIRE(rb.dequeue() == 0);
37:
38:     BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
39: }
40:
41: BOOST_AUTO_TEST_CASE(test1) {
42:     RingBuffer rb(3);
43:     rb.enqueue(2);
44:     rb.enqueue(3);
45:     rb.enqueue(4);
46:     BOOST_REQUIRE(rb.isFull());
47:     BOOST_REQUIRE_THROW(rb.enqueue(5), std::runtime_error)
48: }
```

```
1: /*Copyright [2016] <Albara Mehene> */
2: #ifndef RINGBUFFER_HPP
3: #define RINGBUFFER_HPP
4:
5: #include <boost/test/unit_test.hpp>
6:
7: #include <SFML/Graphics.hpp>
8: #include <SFML/System.hpp>
9: #include <SFML/Audio.hpp>
10: #include <SFML/Window.hpp>
11: #include <stdint.h>
12: #include <iostream>
13: #include <string>
14: #include <exception>
15: #include <stdexcept>
16: #include <vector>
17:
18: class RingBuffer{
19: public:
20:     explicit RingBuffer(int capacity);
21:     int size();
22:     bool isEmpty();
23:     bool isFull();
24:     void enqueue(int16_t x);
25:     int16_t dequeue();
26:     int16_t peek();
27:     void print_out();
28:     ~RingBuffer();
29: private:
30:     int16_t *first;
31:     int16_t *last;
32:     int16_t *array;
33:     int count;
34:     int cap;
35: };
36:
37: #endif
```

```
1: /*Copyright [2016] <Albara Mehene> */
2:
3: #include "RingBuffer.hpp"
4:
5: // creates a empty ringbuffer with a max capacity
6: RingBuffer::RingBuffer(int capacity) {
7:     if(capacity < 1) {
8:         throw std::invalid_argument("capacity must be greater than zero");
9:     }
10:     cap = capacity;
11:     count = 0;
12:     array = new int16_t[capacity];
13:     first = array;
14:     last = array;
15: }
16:
17: // returns the number of items currently in the buffer
18: int RingBuffer::size() {
19:     return count;
20: }
21:
22: // checks to see if the buffer is empty
23: bool RingBuffer::isEmpty() {
24:     if (count == 0) {
25:         return 1;
26:     } else {
27:         return 0;
28:     }
29: }
30:
31: // checks to see if the buffer is full
32: bool RingBuffer::isFull() {
33:     if (count == cap) {
34:         return 1;
35:     } else {
36:         return 0;
37:     }
38: }
39:
40: // add item x to the end of buffer
41: void RingBuffer::enqueue(int16_t x) {
42:     if(isFull() == 1) {
43:         throw std::runtime_error("can't enqueue to a full ring");
44:     }
45:
46:     if (last == (array+(cap-1))) {
47:         count++;
48:         (*last) = x;
49:         last = array;
50:     } else {
51:         count++;
52:         (*last) = x;
53:         last = (last + 1);
54:     }
55: }
56:
57: // deletes and returns the item from the front of the buffer
58: int16_t RingBuffer::dequeue() {
59:     if (isEmpty() == 1) {
60:         throw std::runtime_error("can't dequeue to a empty ring");
61:     }
62: }
```

```
62:
63:     int16_t store;
64:
65:     if(first == (array+(cap-1))){
66:         count--;
67:         store = (*first);
68:         first = array;
69:         return store;
70:     }else{
71:         count--;
72:         store = (*first);
73:         first = (first+1);
74:         return store;
75:     }
76: }
77:
78: // returns item from the front without deleting it
79: int16_t RingBuffer::peek() {
80:     int16_t temp = (*first);
81:     return temp;
82: }
83:
84: RingBuffer::~RingBuffer() {
85:     delete[] array;
86: }
87: void RingBuffer::print_out(){
88:
89:     for(int i = 0; i < cap;i++){
90:         std::cout << "Array:  " << array[i] << std::endl;
91:     }
92: }
```



```
1: /*Copyright Albara Mehene*/
2: #include <SFML/Graphics.hpp>
3: #include <SFML/System.hpp>
4: #include <SFML/Audio.hpp>
5: #include <SFML/Window.hpp>
6:
7: #include <math.h>
8: #include <limits.h>
9:
10: #include <iostream>
11: #include <string>
12: #include <exception>
13: #include <stdexcept>
14: #include <vector>
15:
16: #include "RingBuffer.hpp"
17: #include "GuitarString.hpp"
18:
19: #define SAMPLES_PER_SEC 44100.0
20: #define SAMPLE 37
21:
22: std::vector<sf::Int16> makeSamplesFromString(GuitarString &gs) {
23:     std::vector<sf::Int16> samples;
24:     int duration = 8;
25:     gs.pluck();
26:     int i;
27:     for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
28:         gs.tic();
29:         samples.push_back(gs.sample());
30:     }
31:
32:     return samples;
33: }
34:
35:
36:
37: int main() {
38:     sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Guitar Hero Lite");
39:     sf::Event event;
40:
41:     std::vector < std::vector<sf::Int16> > sample(SAMPLE);
42:     std::vector <sf::Sound> sound(SAMPLE);
43:     std::vector <sf::SoundBuffer> buffer(SAMPLE);
44:     std::string keyboard = ("1234567890qwertyuiopasdfghjklzxcvbnm,");
45:
46:     // inserts all sounds in the buffer
47:     for (int i = 0; i < SAMPLE; i++) {
48:         GuitarString GStemp(440.0 * pow(2, (i - 24)/12.0));
49:         sample[i] = makeSamplesFromString(GStemp);
50:         if (!(buffer[i].loadFromSamples(&(sample[i][0]), sample[i].size(), 2 , 4
4100.0))) {
51:             throw std::runtime_error(" sf::SoundBuffer: failed to load from sample.
");
52:         }
53:         sound[i].setBuffer(buffer[i]);
54:     }
55:
56:     while (window.isOpen()) {
57:         while (window.pollEvent(event)) {
58:             switch (event.type) {
59:                 case sf::Event::Closed:
```

```
60:         window.close();
61:         break;
62:     default:
63:         if (sf::Event::KeyPressed && event.key.code != -1) {
64:             int Key = keyboard.find(event.key.code);
65:             sound[Key].play();
66:         }
67:         break;
68:     }
69:     window.clear();
70:     window.display();
71: }
72: }
73: return 0;
74: }
```

```
1:
2: #define BOOST_TEST_DYN_LINK
3: #define BOOST_TEST_MODULE Main
4: #include <boost/test/unit_test.hpp>
5:
6: #include <vector>
7: #include <exception>
8: #include <stdexcept>
9:
10:
11: #include "GuitarString.hpp"
12:
13: BOOST_AUTO_TEST_CASE(GS) {
14:     std::vector <sf::Int16> v;
15:
16:     v.push_back(0);
17:     v.push_back(2000);
18:     v.push_back(4000);
19:     v.push_back(-10000);
20:
21:     BOOST_REQUIRE_NO_THROW(GuitarString gs = GuitarString(v));
22:
23:     GuitarString gs = GuitarString(v);
24:
25:     // GS is 0 2000 4000 -10000
26:     BOOST_REQUIRE(gs.sample() == 0);
27:
28:     gs.tic();
29:     // it's now 2000 4000 -10000 996
30:     BOOST_REQUIRE(gs.sample() == 2000);
31:
32:     gs.tic();
33:     // it's now 4000 -10000 996 2988
34:     BOOST_REQUIRE(gs.sample() == 4000);
35:
36:     gs.tic();
37:     // it's now -10000 996 2988 -2988
38:     BOOST_REQUIRE(gs.sample() == -10000);
39:
40:     gs.tic();
41:     // it's now 996 2988 -2988 -4483
42:     BOOST_REQUIRE(gs.sample() == 996);
43:
44:     gs.tic();
45:     // it's now 2988 -2988 -4483 1984
46:     BOOST_REQUIRE(gs.sample() == 2988);
47:
48:     gs.tic();
49:     // it's now -2988 -4483 1984 0
50:     BOOST_REQUIRE(gs.sample() == -2988);
51:
52:     // a few more times
53:     gs.tic();
54:     BOOST_REQUIRE(gs.sample() == -4483);
55:     gs.tic();
56:     BOOST_REQUIRE(gs.sample() == 1984);
57:     gs.tic();
58:     BOOST_REQUIRE(gs.sample() == 0);
59: }
```

```
1: /*Copyright [2016] <Albara Mehene> */
2: #ifndef GUITARSTRING_H
3: #define GUITARSTRING_H
4:
5: #include <iostream>
6: #include <string>
7: #include <exception>
8: #include <stdexcept>
9: #include <vector>
10: #include <cmath>
11:
12: #include "RingBuffer.hpp"
13:
14: const double DECAY = 0.996;
15: const double SAMPLING_RATE = 44100;
16:
17: class GuitarString{
18: public:
19:     explicit GuitarString(double frequency);
20:     explicit GuitarString(std::vector <sf::Int16> init);
21:     GuitarString();
22:     void pluck();
23:     void tic();
24:     sf::Int16 sample();
25:     int time();
26:     ~GuitarString();
27: private:
28:     RingBuffer *_rb;
29:     int _ticNum;
30:     int G_cap;
31: };
32:
33: #endif
```

```
1: #include "GuitarString.hpp"
2:
3: /*create a guitar string of the given frequency using a
4: sampling rate of 44,100.*/
5: GuitarString::GuitarString(double frequency){
6:     G_cap = (ceil(SAMPLING_RATE/frequency));
7:
8:     this->_rb = new RingBuffer(G_cap);
9:
10:    //fill with 0s to represent the guitar string at rest
11:    for(int i = 0; i < G_cap; i++){
12:        this->_rb->enqueue(0);
13:    }
14: }
15:
16: /*create a guitar string with size and initial values are given
17: by the vector*/
18: GuitarString::GuitarString(std::vector <sf::Int16> init){
19:
20:     this->_rb = new RingBuffer(init.size());
21:
22:     for (std::vector<sf::Int16>::iterator i = init.begin(); i != init.en
d(); ++i){
23:         _rb->enqueue(*i);
24:
25:     }
26: }
27:
28: GuitarString::GuitarString(){
29:
30: }
31:
32: /*Pluck the guitar string by replacing the buffer with random
33: values, representing white noise*/
34: void GuitarString::pluck(){
35:     //empty the buffer
36:     delete _rb;
37:     _rb = new RingBuffer(G_cap);
38:
39:     //replace with random noise (white noise)
40:     for(int i = 0; i < G_cap; ++i){
41:
42:         _rb->enqueue((int16_t)(rand() & 0xffff));
43:     }
44:
45: }
46:
47: /*advance the simulation one time step */
48: void GuitarString::tic(){
49:     _ticNum++;
50:
51:     double first = _rb->dequeue();
52:     double front = sample();
53:
54:     //std::cout << "Decay: " << DECAY << std::endl;
55:
56:     double value = DECAY * ((first + front)/2);
57:
58:     //std::cout << "value: " << value << std::endl;
59:     _rb->enqueue(value);
60: }
```

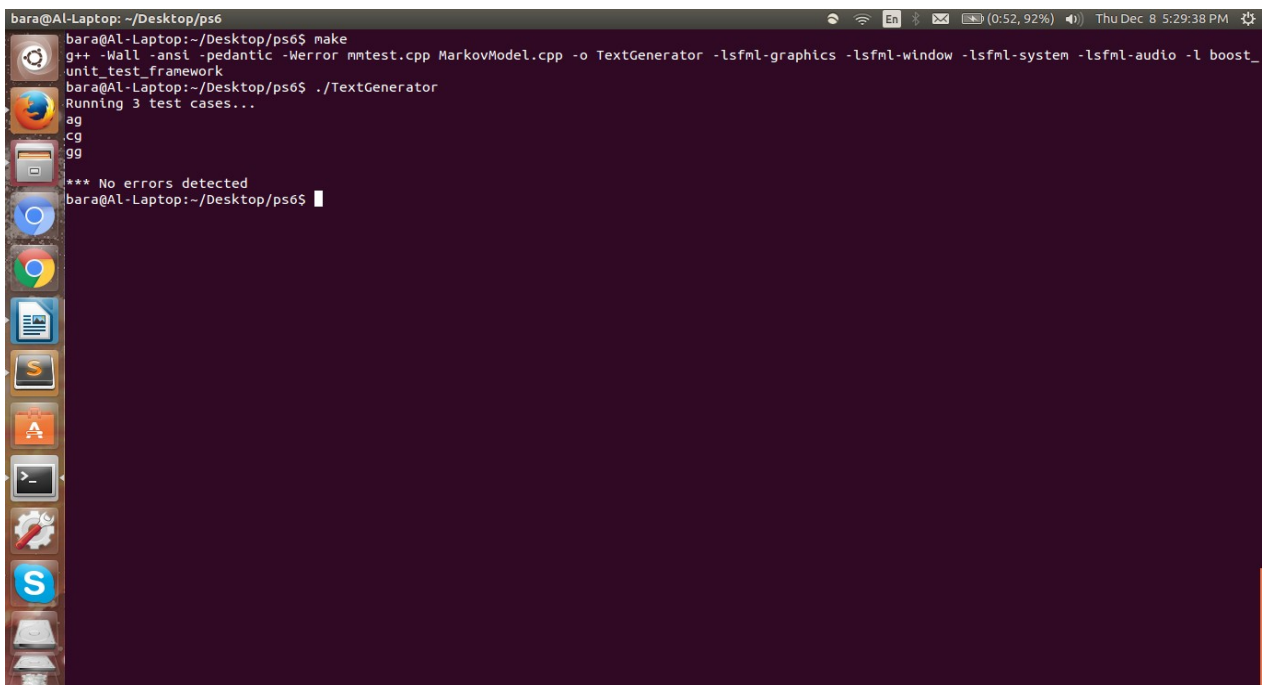
```
61: /*return the current sample*/
62: sf::Int16 GuitarString::sample(){
63:     //May be returning this incorrectly
64:     return _rb->peek();
65: }
66:
67: /*return number of times tic was called so far*/
68: int GuitarString::time(){
69:
70:     return _ticNum;
71: }
72:
73: GuitarString::~GuitarString(){
74:     //need to use delete function
75:     delete _rb;
76:
77: }
```

PS6: Markov Model of Natural Language

The focus on this assignment is to convert the concept of the Markov Model into code. The Markov model is a concept that predicts each letter for the next and finds the probability of what will be the next set of letters. To accomplish this assignment we needed to create a Markov model of order k that are given from a given text of type string. Then using a Markov chain to get the current state in a k -gram of the next character from the selected random probability from Markov model.

One key library we used was the map library. The map function is a container that can store elements formed by a combination of key values and mapped values in any specific order. In the constructor, I saved the string and number of k values to the member variables. Then created a string that took each character and pushed back the letter. I needed to check the char if it's already in the alphabet by having a for loop and iterator through the matrix. Then insert the character into the map. In the `randk` function, I would have a random value from the k gram and then finally print it out. The `gen` function would print out the next k gram until the size of string. Then the operator would be called each time the map needed to be printed out.

Learning about the data type map using to store different data types. I understood how the Markov model could be used in different projects. Thankfully, I had help from a tutor to help explain the layout of this program and what is needed.



```
bara@Al-Laptop: ~/Desktop/ps6
bara@Al-Laptop:~/Desktop/ps6$ make
g++ -Wall -ansi -pedantic -Werror mmtest.cpp MarkovModel.cpp -o TextGenerator -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio -l boost_unit_test_framework
bara@Al-Laptop:~/Desktop/ps6$ ./TextGenerator
Running 3 test cases...
ag
cg
gg
*** No errors detected
bara@Al-Laptop:~/Desktop/ps6$
```

```
1: cc = g++
2:
3: all : MarkovModel
4:
5: MarkovModel : MarkovModel.o mmtest.o
6:      $(cc) -Wall -ansi -pedantic -Werror mmtest.cpp MarkovModel.cpp -o TextGenerator -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio -l boost_unit_test_framework
7:
8: MarkovModel.o : MarkovModel.hpp
9:      $(cc) -c MarkovModel.cpp -o MarkovModel.o
10:
11: mmtest.o : MarkovModel.hpp
12:      $(cc) -c mmtest.cpp -o mmtest.o
13:
14: clean :
15:      rm *.o TextGenerator debug
16:
17: debug :
18:      g++ -g -Wall -ansi -pedantic -Werror mmtest.cpp MarkovModel.cpp -o Markov -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio -l boost_unit_test_framework
```



```
1: /*<Copyright Fred Martain*/
2:
3: #define BOOST_TEST_DYN_LINK
4: #define BOOST_TEST_MODULE Main
5: #include <boost/test/unit_test.hpp>
6:
7: #include <iostream>
8: #include <string>
9: #include <exception>
10: #include <stdexcept>
11:
12: #include "MarkovModel.hpp"
13:
14: // using namespace std;
15:
16: BOOST_AUTO_TEST_CASE(order0) {
17:     // normal constructor
18:     BOOST_REQUIRE_NO_THROW(MarkovModel("gagggagagggcgagaaa", 0));
19:
20:     MarkovModel mm("gagggagagggcgagaaa", 0);
21:
22:     BOOST_REQUIRE(mm.order() == 0);
23:     // length of input in constructor
24:     BOOST_REQUIRE(mm.freq("") == 17);
25:     BOOST_REQUIRE_THROW(mm.freq("x"), std::runtime_error);
26:
27:     BOOST_REQUIRE(mm.freq("", 'g') == 9);
28:     BOOST_REQUIRE(mm.freq("", 'a') == 7);
29:     BOOST_REQUIRE(mm.freq("", 'c') == 1);
30:     BOOST_REQUIRE(mm.freq("", 'x') == 0);
31: }
32:
33: BOOST_AUTO_TEST_CASE(order1) {
34:     // normal constructor
35:     BOOST_REQUIRE_NO_THROW(MarkovModel("gagggagagggcgagaaa", 1));
36:
37:     MarkovModel mm("gagggagagggcgagaaa", 1);
38:
39:     BOOST_REQUIRE(mm.order() == 1);
40:     BOOST_REQUIRE_THROW(mm.freq(""), std::runtime_error);
41:     BOOST_REQUIRE_THROW(mm.freq("xx"), std::runtime_error);
42:
43:     BOOST_REQUIRE(mm.freq("a") == 7);
44:     BOOST_REQUIRE(mm.freq("g") == 9);
45:     BOOST_REQUIRE(mm.freq("c") == 1);
46:
47:     BOOST_REQUIRE(mm.freq("a", 'a') == 2);
48:     BOOST_REQUIRE(mm.freq("a", 'c') == 0);
49:     BOOST_REQUIRE(mm.freq("a", 'g') == 5);
50:
51:     BOOST_REQUIRE(mm.freq("c", 'a') == 0);
52:     BOOST_REQUIRE(mm.freq("c", 'c') == 0);
53:     BOOST_REQUIRE(mm.freq("c", 'g') == 1);
54:
55:     BOOST_REQUIRE(mm.freq("g", 'a') == 5);
56:     BOOST_REQUIRE(mm.freq("g", 'c') == 1);
57:     BOOST_REQUIRE(mm.freq("g", 'g') == 3);
58:
59:     BOOST_REQUIRE_NO_THROW(mm.randk("a"));
60:     BOOST_REQUIRE_NO_THROW(mm.randk("c"));
61:     BOOST_REQUIRE_NO_THROW(mm.randk("g"));
```

```
62:
63:     BOOST_REQUIRE_THROW(mm.randk("x"), std::runtime_error);
64:
65:     BOOST_REQUIRE_THROW(mm.randk("xx"), std::runtime_error);
66: }
67:
68: BOOST_AUTO_TEST_CASE(order2) {
69:     // normal constructor
70:     BOOST_REQUIRE_NO_THROW(MarkovModel("gagggagagggcgagaaa", 2));
71:
72:     MarkovModel mm("gagggagagggcgagaaa", 2);
73:
74:     BOOST_REQUIRE(mm.order() == 2);
75:
76:     BOOST_REQUIRE_THROW(mm.freq(""), std::runtime_error);
77:     BOOST_REQUIRE_THROW(mm.freq("x"), std::runtime_error);
78:     BOOST_REQUIRE_NO_THROW(mm.freq("xx"));
79:     // kgram is wrong length
80:     BOOST_REQUIRE_THROW(mm.freq("", 'g'), std::runtime_error);
81:     // kgram is wrong length
82:     BOOST_REQUIRE_THROW(mm.freq("x", 'g'), std::runtime_error);
83:     // kgram is wrong length
84:     BOOST_REQUIRE_THROW(mm.freq("xxx", 'g'), std::runtime_error);
85:
86:
87:     BOOST_REQUIRE(mm.freq("aa") == 2);
88:     BOOST_REQUIRE(mm.freq("aa", 'a') == 1);
89:     BOOST_REQUIRE(mm.freq("aa", 'c') == 0);
90:     BOOST_REQUIRE(mm.freq("aa", 'g') == 1);
91:
92:     BOOST_REQUIRE(mm.freq("ag") == 5);
93:     BOOST_REQUIRE(mm.freq("ag", 'a') == 3);
94:     BOOST_REQUIRE(mm.freq("ag", 'c') == 0);
95:     BOOST_REQUIRE(mm.freq("ag", 'g') == 2);
96:
97:     BOOST_REQUIRE(mm.freq("cg") == 1);
98:     BOOST_REQUIRE(mm.freq("cg", 'a') == 1);
99:     BOOST_REQUIRE(mm.freq("cg", 'c') == 0);
100:    BOOST_REQUIRE(mm.freq("cg", 'g') == 0);
101:
102:    BOOST_REQUIRE(mm.freq("ga") == 5);
103:    BOOST_REQUIRE(mm.freq("ga", 'a') == 1);
104:    BOOST_REQUIRE(mm.freq("ga", 'c') == 0);
105:    BOOST_REQUIRE(mm.freq("ga", 'g') == 4);
106:
107:    BOOST_REQUIRE(mm.freq("gc") == 1);
108:    BOOST_REQUIRE(mm.freq("gc", 'a') == 0);
109:    BOOST_REQUIRE(mm.freq("gc", 'c') == 0);
110:    BOOST_REQUIRE(mm.freq("gc", 'g') == 1);
111:
112:    BOOST_REQUIRE(mm.freq("gg") == 3);
113:    BOOST_REQUIRE(mm.freq("gg", 'a') == 1);
114:    BOOST_REQUIRE(mm.freq("gg", 'c') == 1);
115:    BOOST_REQUIRE(mm.freq("gg", 'g') == 1);
116: }
```

```
1: /* <Copyright Abara Mehene*/
2: #ifndef MARKOV_MODEL_HPP
3: #define MARKOV_MODEL_HPP
4:
5: #include <iostream>
6: #include <map>
7: #include <string>
8: #include <stdexcept>
9: #include <algorithm>
10:
11: class MarkovModel {
12: private:
13:     int _order;
14:     std::map<std::string, int> _kgrams; // must #include <map>
15:     std::string _alphabet;
16:     // space
17: public:
18:     // create a Markov model of order k from given text
19:     // Assume that text has length at least k.
20:     MarkovModel(std::string text, int k);
21:
22:     // order k of Markov model
23:     int order();
24:
25:     // number of occurrences of kgram in text
26:     // (throw an exception if kgram is not of length k)
27:     int freq(std::string kgram);
28:
29:     // number of times that character c follows kgram
30:     // if order=0, return num of times char c appears
31:     // (throw an exception if kgram is not of length k)
32:     int freq(std::string kgram, char c);
33:
34:     // random character following given kgram
35:     // (Throw an exception if kgram is not of length k.
36:     // Throw an exception if no such kgram.)
37:     char randk(std::string kgram);
38:
39:     // generate a string of length T characters
40:     // by simulating a trajectory through the corresponding
41:     // Markov chain. The first k characters of the newly
42:     // generated string should be the argument kgram.
43:     // Throw an exception if kgram is not of length k.
44:     // Assume that T is at least k.
45:     std::string gen(std::string kgram, int T);
46:
47:     // overload the stream insertion operator and display
48:     // the internal state of the Markov Model. Print out
49:     // the order, the alphabet, and the frequencies of
50:     // the k-grams and k+1-grams.
51:     friend std::ostream& operator<<(std::ostream &out, MarkovModel &mm);
52:
53:     ~MarkovModel();
54: };
55:
56: #endif
```

```
1: /* <Copyright Abara Mehene*/
2:
3: #include "MarkovModel.hpp"
4: #include <utility>
5: #include <map>
6: #include <string>
7:
8: MarkovModel::MarkovModel(std::string text, int k) {
9:     _order = k;
10:    _alphabet = text;
11:
12:    char character;
13:    bool value = 0;
14:    std::string circularString = text;
15:    std::string tempString;
16:
17:    // INSERTING CHARS INTO THE ALPHABET
18:
19:    // store the chars that appear in text into the alphabet
20:    for (int i = 0; i < _order; ++i) {
21:        // insert the new char into the alphabet
22:        circularString.push_back(text[i]);
23:    }
24:
25:    // check if the char in the text is already in the alphabet
26:    for (unsigned int i = 0; i < text.length(); ++i) {
27:        character = text.at(i);
28:        value = 0;
29:        // check if we already have the char
30:        for (unsigned int j = 0; j < _alphabet.length(); ++j) {
31:            if (_alphabet.at(j) == character)
32:                value = 1;
33:        }
34:        // do we store the char into the alphabet?
35:        if (!value)
36:            _alphabet.push_back(character);
37:    }
38:
39:
40:    std::map<std::string, int>::iterator it;
41:    int temp_count = 0;
42:
43:    // get a substring from the text and inserting it into kgram
44:    for (int x = _order; x <= _order + 1; ++x) {
45:        for (unsigned int y = 0; y < text.length(); ++y) {
46:            tempString = circularString.substr(y, x);
47:            _kgrams.insert(std::pair<std::string, int>(tempString, 0));
48:            it = _kgrams.find(tempString);
49:            temp_count = it->second;
50:            temp_count++;
51:            _kgrams[tempString] = temp_count;
52:        }
53:    }
54: }
55:
56: // returns order
57: int MarkovModel::order() {
58:     return _order;
59: }
60:
61: int MarkovModel::freq(std::string kgram) {
```

```
62: // error check
63: if ((unsigned)_order != kgram.length())
64:     throw std::runtime_error("Kgram is not of length k");
65: // space
66: std::map<std::string, int>::iterator numOfkGram;
67: // go through the map and count how many
68: // times we have the string kgram
69: numOfkGram = _kgrams.find(kgram);
70:
71: // return the kgram we find
72: if (numOfkGram == _kgrams.end())
73:     return 0;
74: return numOfkGram->second;
75: }
76:
77: int MarkovModel::freq(std::string kgram, char c) {
78:     // error check
79:     if (kgram.length() != (unsigned)_order)
80:         throw std::runtime_error("Kgram is not of length k");
81:
82:     // put c into kgram then find the new kgram
83:     std::map<std::string, int>::iterator numOfkGram;
84:     kgram.push_back(c);
85:     numOfkGram = _kgrams.find(kgram);
86:
87:     // if there is the kgram
88:     if (numOfkGram == _kgrams.end())
89:         return 0;
90:     return numOfkGram->second;
91: }
92:
93: char MarkovModel::randk(std::string kgram) {
94:     unsigned int seed = time(NULL);
95:     int randomValue;
96:     std::string randomInput;
97:
98:     // error check
99:     if (kgram.length() != (unsigned)_order)
100:         throw std::runtime_error("Kgram is not of length k");
101:     // space
102:     // try to find the kgram
103:     std::map<std::string, int>::iterator temp;
104:     temp = _kgrams.find(kgram);
105:
106:     int kgram_freq = freq(kgram);
107:
108:     // if there is no such kgram
109:     if (temp == _kgrams.end())
110:         throw std::runtime_error("No such kgram");
111:
112:     for (;;) {
113:         // gets random value from the kgram_freq
114:         randomValue = rand_r(&seed) % kgram_freq;
115:         randomInput = kgram + _alphabet[randomValue];
116:         // if we are at the end return a random char
117:         if (temp != _kgrams.end()) {
118:             std::cout << randomInput << std::endl;
119:             return _alphabet[randomValue];
120:         }
121:     }
122: }
```

```
123:
124: std::string MarkovModel::gen(std::string kgram, int T) {
125:     // error check
126:     if (kgram.length() != (unsigned)_order)
127:         throw std::runtime_error("Kgram is not of length k");
128:
129:     // put the initial kgram into our string
130:     std::string tempkGram = kgram;
131:
132:     // append the random character to the end of our output
133:     // until size the string is of size T
134:     for (int i = kgram.length(); i < T; ++i) {
135:         tempkGram.push_back(randk(kgram));
136:     }
137:
138:     return tempkGram;
139: }
140:
141: std::ostream& operator<<(std::ostream &out, MarkovModel &mm) {
142:     std::map<std::string, int>::iterator it;
143:     // basic output
144:     out << "Order: " << mm._order << std::endl;
145:     out << "Alphabet: " << mm._alphabet << std::endl;
146:     out << "Kgrams map: " << std::endl;
147:
148:     for (it == mm._kgrams.begin(); it != mm._kgrams.end(); it++) {
149:         out << it->first << it->second << std::endl;
150:     }
151:     return out;
152: }
153:
154: MarkovModel::~MarkovModel() {
155:     // destructor
156: }
```

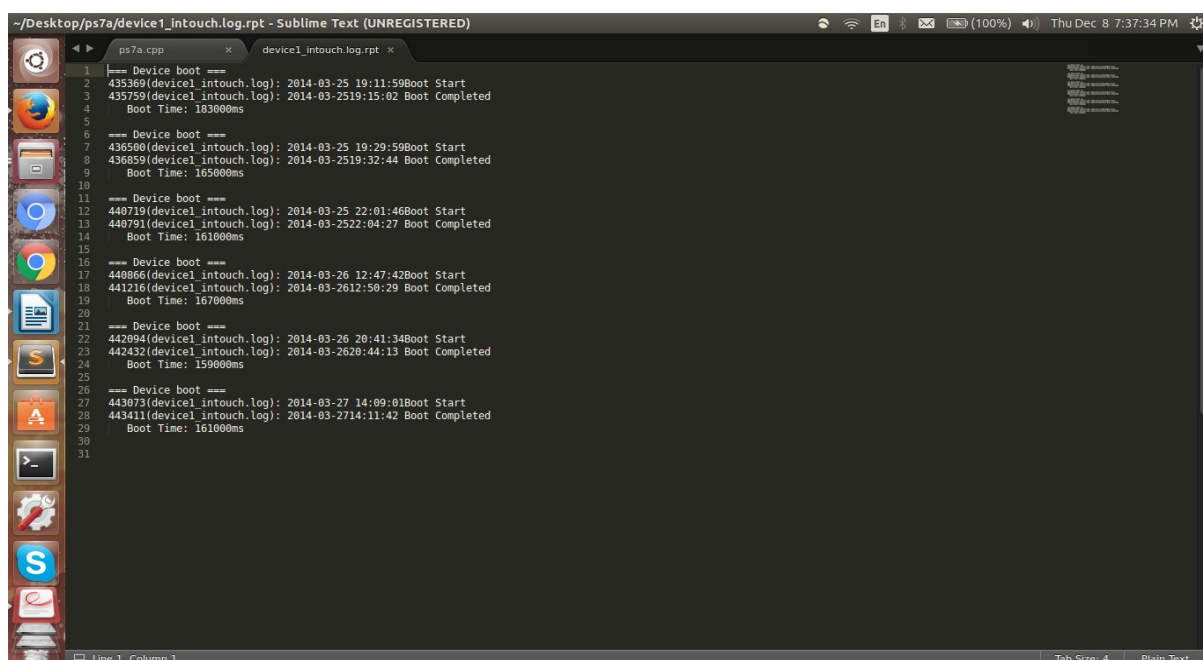
PS7: Kronos Intouch Parsing

PS7a:

The Kronos Intouch was a interesting assignment to work on. The objective of this assignment was to implement a scan that can go through the inTouch log files finding specific lines and outputting them into a output file. To accomplish this, we needed to learn about regex commands. The assignment wanted us to find a specific line that started with *(log.c.166) server started* and to find the completion of the boot. We needed to find the elapsed time in between the start time and end time.

One library that was needed to accomplish this assignment was the boost regex software. This software was to have the ability to use functional regex commands and use functions that could recognize them. To start, I to use the *using* function to shorten the deceleration type it make it easier. I created regex commands for the start line and end line. I went through each line and checked each time. If the start exits but the end doesn't, it will give a incomplete error. After it completes the loop, I would subtract the end by the start to get the total elapsed time.

After accomplishing this assignment, I learned how to use regex commands and how to use they can be useful in the future commands. Formating the output file to look specific a specific output was a great skill when getting a job. Also, knowing the what functions are from the boost regex library was also a plus.



```
--/Desktop/ps7a/device1_intouch.log.rpt - Sublime Text (UNREGISTERED)
ps7a.cpp
device1_intouch.log.rpt
1  === Device boot ===
2  435369(device1_intouch.log): 2014-03-25 19:11:59Boot Start
3  435759(device1_intouch.log): 2014-03-2519:15:02 Boot Completed
4  Boot Time: 183000ms
5
6  === Device boot ===
7  436500(device1_intouch.log): 2014-03-25 19:29:59Boot Start
8  436859(device1_intouch.log): 2014-03-2519:32:44 Boot Completed
9  Boot Time: 165000ms
10
11 === Device boot ===
12 440719(device1_intouch.log): 2014-03-25 22:01:46Boot Start
13 440791(device1_intouch.log): 2014-03-2522:04:27 Boot Completed
14 Boot Time: 161000ms
15
16 === Device boot ===
17 440866(device1_intouch.log): 2014-03-26 12:47:42Boot Start
18 441216(device1_intouch.log): 2014-03-2612:50:29 Boot Completed
19 Boot Time: 167000ms
20
21 === Device boot ===
22 442094(device1_intouch.log): 2014-03-26 20:41:34Boot Start
23 442432(device1_intouch.log): 2014-03-2620:44:13 Boot Completed
24 Boot Time: 159000ms
25
26 === Device boot ===
27 443073(device1_intouch.log): 2014-03-27 14:09:01Boot Start
28 443411(device1_intouch.log): 2014-03-2714:11:42 Boot Completed
29 Boot Time: 161000ms
30
31
```

```
1: CC = g++
2: CFLAGS = -Wall -Werror -ansi -pedantic -std=c++0x
3: BOOST = -lboost_regex
4:
5: all: ps7a
6:
7: ps7a: ps7a.o
8:      $(CC) ps7a.o -o ps7a $(BOOST)
9:
10: ps7a.o:
11:      $(CC) -c ps7a.cpp $(CFLAGS)
12:
13: clean:
14:      rm -f *.o *~ ps7a
```



```
1: // <Copyright Owners Albara Mehene & Sean Nishi>
2: // regex_match example
3: #include <boost/regex.hpp>
4: #include <boost/date_time/gregorian/gregorian.hpp>
5: #include <boost/date_time/posix_time/posix_time.hpp>
6: #include <iostream>
7: #include <string>
8: #include <fstream>
9:
10: using boost::gregorian::date;
11: using boost::gregorian::years;
12: using boost::gregorian::months;
13: using boost::gregorian::days;
14: using boost::gregorian::date_duration;
15: using boost::gregorian::date_period;
16: using boost::gregorian::from_simple_string;
17:
18: using boost::posix_time::ptime;
19: using boost::posix_time::hours;
20: using boost::posix_time::minutes;
21: using boost::posix_time::seconds;
22: using boost::posix_time::time_duration;
23:
24:
25: int main(int argc, char* argv[]) {
26:     if (argc != 2) {
27:         std::cout << "ERROR: input only one file" << std::endl;
28:         return -1;
29:     }
30:
31:     // open the input file
32:     std::ifstream logFile;
33:     logFile.open(argv[1]);
34:     // name of file
35:     std::string logName(argv[1]);
36:     std::string outputName = logName + ".rpt";
37:     // create the output file
38:     std::ofstream outputFile;
39:     outputFile.open(outputName.c_str());
40:     // space
41:     std::string line;
42:     date stored_date;
43:     date finished_date;
44:     ptime beginTime;
45:     ptime endTime;
46:     boost::smatch m;
47:     time_duration total_time;
48:     // space
49:     bool s_boot = false;
50:     int lineNum = 1;
51:
52:     // Start of boot: 2014-02-01 14:02:32: (log.c.166) server started
53:     boost::regex Boot_Start(
54:         "([0-9]{4})-([0-9]{2})-([0-9]{2}) "
55:         "([0-9]{2}):([0-9]{2}):([0-9]{2}): "
56:         "\\(log.c.166\\) server started.*");
57:     // If we find the text:
58:     // "2014-01-26 09:58:04.362:INFO:oejs.AbstractConnector:Started
59:     // SelectChannelConnector@0.0.0.0:9080"
60:     boost::regex Boot_End(
61:         "([0-9]{4})-([0-9]{2})-([0-9]{2}) "
```

```
62:      "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}):INFO:"
63:      "oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080.*");
64:  // check if input file is open
65:  if (!logFile.is_open()) {
66:      std::cout << "ERROR: no input log file" << std::endl;
67:      return -1;
68:  } else {
69:      // Go through the loop
70:      while (getline(logFile, line)) {
71:          // search the start regex code
72:          if (regex_search(line, m, Boot_Start)) {
73:              // store them into date and time
74:              stored_date = date(stoi(m[1]), stoi(m[2]), stoi(m[3]));
75:              beginTime = ptime(stored_date,
76:                               time_duration(stoi(m[4]), stoi(m[5]), stoi(m[6])));
77:              // condition to see if it will fail to go to the next condition
78:              if (s_boot) {
79:                  s_boot = false;
80:                  outputFile << "**** Incomplete boot ****\n" << std::endl;
81:              }
82:              // draw the start into the output file
83:              outputFile << "=== Device boot ===\n"
84:                  << lineNumber << "(" << logName << "): "
85:                  << m[1] << "-" << m[2] << "-" << m[3]
86:                  << " "
87:                  << m[4] << ":" << m[5] << ":" << m[6]
88:                  << "Boot Start" << std::endl;
89:              s_boot = true;
90:              // Then do the same to the end but checking the regex
91:          } else if (regex_search(line, m, Boot_End)) {
92:              finished_date = date(stoi(m[1]), stoi(m[2]), stoi(m[3]));
93:              endTime = ptime(stored_date,
94:                              time_duration(stoi(m[4]), stoi(m[5]), stoi(m[6])));
95:              // output the rest
96:              outputFile << lineNumber << "(" << logName << "): "
97:                  << m[1] << "-" << m[2] << "-" << m[3]
98:                  << m[4] << ":" << m[5] << ":" << m[6]
99:                  << " Boot Completed" << std::endl;
100:              total_time = endTime - beginTime;
101:              outputFile << " "
102:                  << "Boot Time: " << total_time.total_milliseconds()
103:                  << "ms\n" << std::endl;
104:              s_boot = false;
105:          }
106:          // increment line number to go to next line
107:          lineNumber++;
108:      }
109:      // closing the files
110:      logFile.close();
111:      outputFile.close();
112:  }
113:  return 0;
114: }
```

PS7b & Psx:

This part of the assignment, we now needed to print out the services in the program to the output. Any time a service, has the line *Starting Service. Logging 1.0* , the program will save that line and print them out into the output file. The program needs to print out the previous problem first then would print out the services after. In addition to accomplishing this assignment, finding upgrades and downgrades to software in logs need to be included.

Key concepts to this assignment is like the previous problem but now to print out additional conditions. Learning how to go back to the start of the boot and finding all lines with services and upgrades/downgrades.

I did not learn so much from this assignment. Sadly I could not finish this program and was not able to takeaway anything. Because of the previous part, I was able to write down the correct regex commands. Though I could not figure out how to go back to the starting line and print out the services.

```
1: CC = g++
2: CFLAGS = -Wall -Werror -ansi -pedantic -std=c++0x
3: BOOST = -lboost_regex
4:
5: all: ps7b
6:
7: ps7b: ps7b.o
8:      $(CC) ps7b.o -o ps7b $(BOOST)
9:
10: ps7b.o:
11:      $(CC) -c ps7b.cpp $(CFLAGS)
12:
13: clean:
14:      rm -f *.o *~ ps7b
```

```
1: // <Copyright Owners Albara Mehene & Sean Nishi>
2: // regex_match example
3: #include <boost/regex.hpp>
4: #include <boost/date_time/gregorian/gregorian.hpp>
5: #include <boost/date_time/posix_time/posix_time.hpp>
6: #include <iostream>
7: #include <string>
8: #include <fstream>
9:
10: using boost::gregorian::date;
11: using boost::gregorian::years;
12: using boost::gregorian::months;
13: using boost::gregorian::days;
14: using boost::gregorian::date_duration;
15: using boost::gregorian::date_period;
16: using boost::gregorian::from_simple_string;
17:
18: using boost::posix_time::ptime;
19: using boost::posix_time::hours;
20: using boost::posix_time::minutes;
21: using boost::posix_time::seconds;
22: using boost::posix_time::time_duration;
23:
24:
25: int main(int argc, char* argv[]) {
26:     if (argc != 2) {
27:         std::cout << "ERROR: input only one file" << std::endl;
28:         return -1;
29:     }
30:
31:     // open the input file
32:     std::ifstream logFile;
33:     logFile.open(argv[1]);
34:     // name of file
35:     std::string logName(argv[1]);
36:     std::string outputName = logName + ".rpt";
37:     // create the output file
38:     std::ofstream outputFile;
39:     outputFile.open(outputName.c_str());
40:     // space
41:     std::string line;
42:     date stored_date;
43:     date finished_date;
44:     ptime beginTime;
45:     ptime endTime;
46:     boost::smatch m;
47:     time_duration total_time;
48:     // space
49:     bool s_boot = false;
50:     int lineNum = 1;
51:
52:     // Start of boot: 2014-02-01 14:02:32: (log.c.166) server started
53:     boost::regex Boot_Start(
54:         "([0-9]{4})-([0-9]{2})-([0-9]{2}) "
55:         "([0-9]{2}):([0-9]{2}):([0-9]{2}): "
56:         "\\(log.c.166\\) server started.*");
57:     // If we find the text:
58:     // "2014-01-26 09:58:04.362:INFO:oejs.AbstractConnector:Started
59:     // SelectChannelConnector@0.0.0.0:9080"
60:     boost::regex Boot_End(
61:         "([0-9]{4})-([0-9]{2})-([0-9]{2}) "
```

```
62:      "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}):INFO:"
63:      "oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080.*");
64:      // Space
65:      boost::regex start_service(
66:          "Starting\\ Service\\.\\.\\.\\. ([a-z]|[A-Z]+).+");
67:      boost::regex end_service(
68:          "Service\\.\\. started\\.\\. successfully\\.\\.\\.\\. "
69:          "([a-z]|[A-Z]+).+\\.\\.\\.\\.([0-9]+).+");
70:      // check if input file is open
71:      if (!logFile.is_open()) {
72:          std::cout << "ERROR: no input log file" << std::endl;
73:          return -1;
74:      } else {
75:          // Go through the loop
76:          while (getline(logFile, line)) {
77:              // search the start regex code
78:              if (regex_search(line, m, Boot_Start)) {
79:                  // store them into date and time
80:                  stored_date = date(stoi(m[1]), stoi(m[2]), stoi(m[3]));
81:                  beginTime = ptime(stored_date,
82:                      time_duration(stoi(m[4]), stoi(m[5]), stoi(m[6])));
83:                  // condition to see if it will fail to go to the next condition
84:                  if (s_boot) {
85:                      s_boot = false;
86:                      outputFile << "**** Incomplete boot ****\n" << std::endl;
87:                  }
88:                  // draw the start into the output file
89:                  outputFile << "=== Device boot ===\n"
90:                      << lineNumber << "(" << logName << "): "
91:                      << m[1] << "-" << m[2] << "-" << m[3]
92:                      << " "
93:                      << m[4] << ":" << m[5] << ":" << m[6]
94:                      << "Boot Start" << std::endl;
95:                  s_boot = true;
96:                  // Then do the same to the end but checking the regex
97:              } else if (regex_search(line, m, Boot_End)) {
98:                  finished_date = date(stoi(m[1]), stoi(m[2]), stoi(m[3]));
99:                  endTime = ptime(stored_date,
100:                      time_duration(stoi(m[4]), stoi(m[5]), stoi(m[6])));
101:                  // output the rest
102:                  outputFile << lineNumber << "(" << logName << "): "
103:                      << m[1] << "-" << m[2] << "-" << m[3]
104:                      << m[4] << ":" << m[5] << ":" << m[6]
105:                      << " Boot Completed" << std::endl;
106:                  total_time = endTime - beginTime;
107:                  outputFile << " "
108:                      << "Boot Time: " << total_time.total_milliseconds()
109:                      << "ms\n" << std::endl;
110:                  s_boot = false;
111:              }
112:              // increment line number to go to next line
113:              lineNumber++;
114:          }
115:          // closing the files
116:          logFile.close();
117:          outputFile.close();
118:      }
119:      return 0;
120: }
```