Jason Downing
91.204.202 Project Portfolio
May 1st, 2015

Contents:

PS0: Hello World

The Assignment

Hello World was our first Computing IV assignment. The main goal of this assignment was to setup our Linux build environment and to test out the SFML audio/graphics library. This included getting Linux running – either through a Virtualbox image or natively, and running some SFML example code to test out SFML. We then had to extend the demo code to make it do something interesting. I was already familiar with Linux at this point, so it really didn't take very long to setup my environment – a few sudo apt-get install commands and I had SFML ready to test out.

Key Concepts

As this was just first assignment, I did not really use any algorithms to create the program. Mostly just basic programming stuff like variables, objects, a couple of while / for loops for keeping the window open and for moving the sprites around. The main idea of this assignment after all was just to test out SFML's different classes, which included Images, Sprites, Text, Texture, and Keyboard. The Texture / Image / Sprite classes also interact with each other – you must load an image into a Texture, and a sprite must set a Texture to be used.

For this assignment, I created a program that uses SFML's sprite class to display a moving image on an SFML window. I added some basic controls to move it around – arrow keys for up/down/left/right, the plus key (+) to increase the size of the sprite and the minus key (-) decreases the size of the image. I also messed around with a few other parts of SFML – I got text to display on the screen, and made a second image move around in a set pattern.

What I Learned

I learned a few things about SFML in this assignment. A few things I learned were: how to use SFML at a basic level, how to display images in an SFML window and even how to control a sprites using SFML's Keyboard library. I did not learn much about Linux, or how to setup a build environment for Linux. This is because I have used Linux previously in a Computing II lab (Prof. Adams), and I have also used Linux in a research lab (Prof. Martin's ECG lab). Personally I also dual-boot Windows and Linux as well, and wrote all of my Computing III C++ programs in Linux – so things like setting up SFML were simple since all I needed to do was boot Linux up and run sudo apt-get install.

Screenshots

Source code for PS0 Hello World

NOTE: For this first assignment, there was no requirement for any header files (.hpps) or implementation files (other .cpps). As such, just the Makefile and Main.cpp are listed.

Makefile

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

Main.cpp

```cpp
#include <iostream>
#include <SFML/Graphics.hpp>
#include <string>

using namespace std;
using namespace sf;

int main()
{
  // Set the size of the window.
  // In this case I make it 605 by 605. The title is "Program 0".
  RenderWindow window(VideoMode(605, 605), "Program 0: Doge Sprite Moves");

  // Change the framerate to make it easier to see the image moving.
  window.setFramerateLimit(1);

  // Loads a doge from file - in this case a doge
  Texture image_texture;

  // Check to see if the file loaded correctly.
  // I make just the doge part of the image load - it's actually 300 by 372
  // but I cut off 50 pixels from the height.
  if (!image_texture.loadFromFile( "sprite.png", IntRect(0, 50, 300, 300)))
  {
    // Return failure if the image doesn't load
    cout << "Failed to load image!\n";
    return EXIT_FAILURE;
  }

  // Make the image look nice.
  image_texture.setSmooth(true);

  // Now set the doge using the texture
  Sprite doge;
  doge.setTexture(image_texture);

  // Set the origin for rotation to the center of the doge.
  doge.setOrigin(150, 125);

  // Move to the top left corner.
  doge.move(150, 100);    // Top left corner

  // Create an image to move in a rectangle pattern
  Texture image2_texture;

  // Load the image
  if (!image2_texture.loadFromFile( "doge.png"))
  {
    // Return failure if the image doesn't load
    cout << "Failed to load image!\n";
    return EXIT_FAILURE;
  }

  image2_texture.setSmooth(true);

```

```
56      // Second doge settings
57      Sprite doge2;
58      doge2.setTexture(image2_texture);  // Set the texture
59      doge2.scale(.25f, .25f);           // Shrink the image
60      doge2.setOrigin(732, 620);         // Set the center of the image
61      doge2.move(300, 300);              // Move the image to the top left corner
62
63      int dir = 1;
64
65      // Create a graphical text to display
66      Font font;
67      if (!font.loadFromFile("arial.ttf"))
68      {
69        cout << "Failed to load font!\n";
70        return EXIT_FAILURE;
71      }
72
73      // Create a text.
74      Text text("Woof", font, 70);
75
76      // Move the text to the bottom center (roughly)
77      text.move(230,500);
78
79      // Window loop
80      while (window.isOpen())
81      {
82        // Process events
83        Event event;
84
85        while(window.pollEvent(event))
86        {
87          // Close window : exit
88          if (event.type == Event::Closed)
89          {
90            window.close();
91          }
92
93          // Move the image if an arrow key is pressed.
94          else if(Event::KeyPressed)
95          {
96            // Arrow keys will move the doge in the
97            // expected direction.
98            if (Keyboard::isKeyPressed(Keyboard::Left))
99            {
100             doge2.move(-15, 0);
101           }
102           else if (Keyboard::isKeyPressed(Keyboard::Right))
103           {
104             doge2.move(15, 0);
105           }
106           else if (Keyboard::isKeyPressed(Keyboard::Up))
107           {
108             doge2.move(0, -15);
109           }
110           else if (Keyboard::isKeyPressed(Keyboard::Down))
111           {
112             doge2.move(0, 15);
```

```
113           }
114
115           // Enter will rotate the doge
116           else if (Keyboard::isKeyPressed(Keyboard::Space))
117           {
118             doge2.rotate(45);
119           }
120
121           // Pressing + will increase the size of the doge.
122           else if (Keyboard::isKeyPressed(Keyboard::Add))
123           {
124             doge2.scale(1.05f, 1.05f);
125           }
126
127           // Pressing - will decrease the size of the doge.
128           else if (Keyboard::isKeyPressed(Keyboard::Subtract))
129           {
130             doge2.scale(.95f, .95f);
131           }
132
133           // Pressing escape will quit the program.
134           else if (Keyboard::isKeyPressed(Keyboard::Escape))
135           {
136             return 0;
137           }
138         }
139       }
140
141       // Clear the screen - so that the doge's previous image gets erased.
142       window.clear();
143
144       // Ifs that will keep the Doge sprite moving in a rectangle pattern.
145       // Basically they increase int dir by 1 until it reaches 5, then it
146       // resets the int dir to 1.
147       switch(dir)
148       {
149         case 1:
150           doge.move(300, 0);      // Top right corner
151           dir++;
152           break;
153
154         case 2:
155           doge.move(0, 325);      // Bottom right corner
156           dir++;
157           break;
158
159         case 3:
160           doge.move(-300, 0);     // Bottom left corner
161           dir++;
162           break;
163
164         case 4:
165           doge.move(0, -325);     // Top left corner again.
166           dir++;
167           break;
168
169         default:
```

```
170             dir = 1;
171             break;
172         }
173 
174         // Redraw the doge
175         window.draw(doge);
176 
177         // This is the second doge - the one that moves in a rectangle.
178         window.draw(doge2);
179 
180         // Draw the string
181         window.draw(text);
182 
183         // Update the window
184         window.display();
185     }
186 
187     return 0;
188 }
```

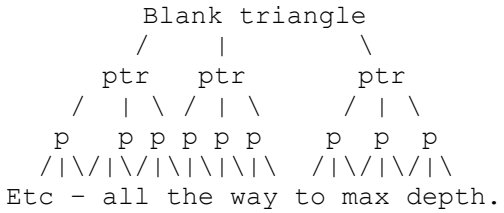
PS1: Recursive Graphic

The Assignment

For this second assignment, we were tasked with implementing the Sierpinski triangle assignment from Princeton. The main idea behind the assignment was to use recursion to create a complex looking triangle that in reality takes only a little bit of code to generate. The main program was to take an integer, N, which would be used to control the depth of the recursion. Our program would then draw one triangle at depth 1, four triangles at depth 2 and so – in effect drawing triangles within each other recursively. A second part of the assignment was to create our own recursively image – different then the Sierpinski triangle design, but still using recursion to generate a cool looking image. I was able to create both the Sierpinski triangle design and my own circle within a circle (which changes colors) design.

Key Concepts

Key to this program was the idea of recursion. I had to figure out a way to implement the triangle drawing within each other recursively, and do so without using a ton of resources. I found the best way to do this was using pointers to other triangles – the first main triangle has three Sierpinski pointers, and those also have pointers to three more triangles and so on until the max depth is reached. I recreated a little diagram in my README file that illustrates this quite well:



By using pointers, I was able to recursively draw the triangles out in the Sierpinski::Draw method – I just did something like:

```
150     _triangle1->draw(target, states);
151     _triangle2->draw(target, states);
152     _triangle3->draw(target, states);
```

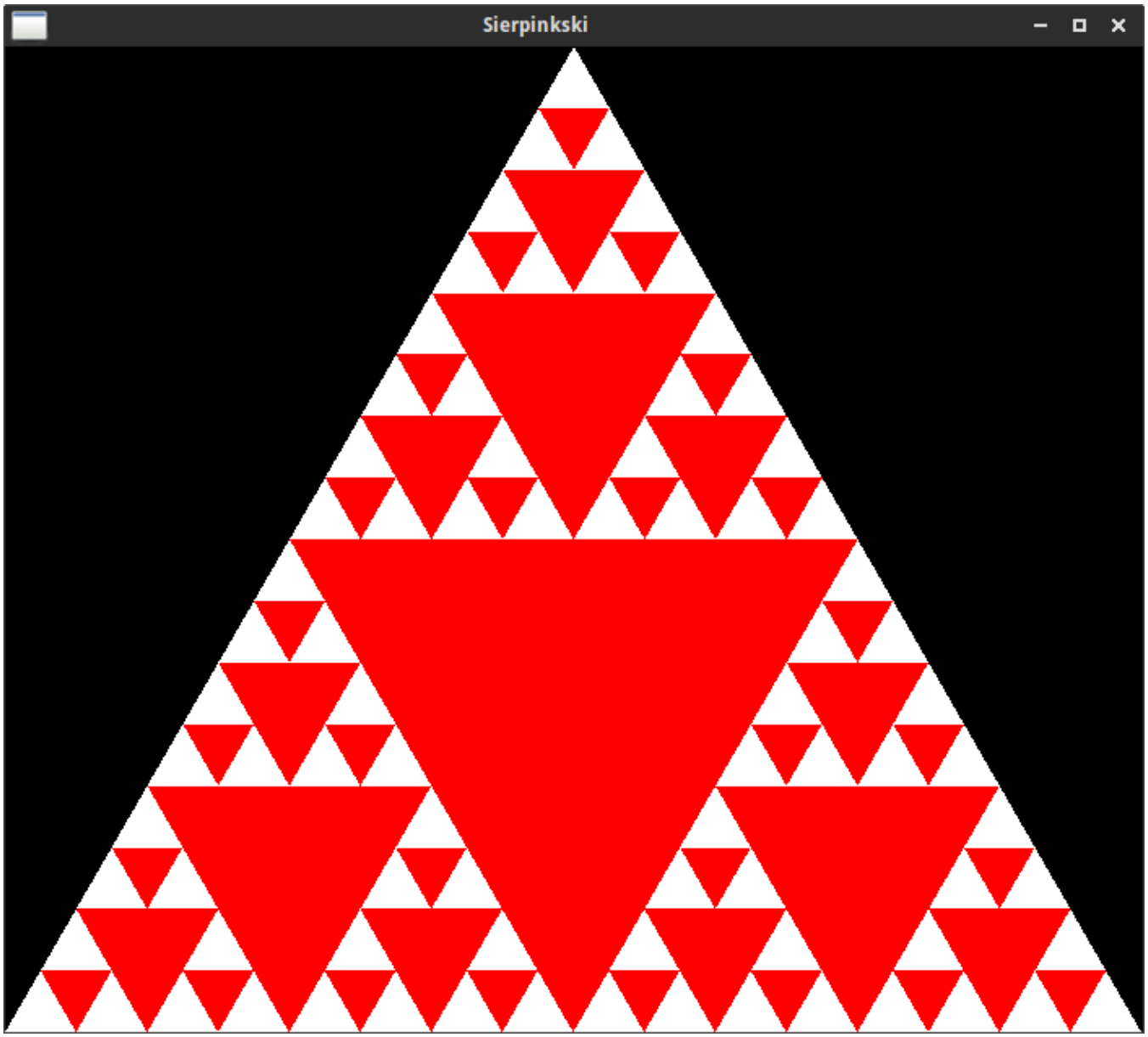
This ends up calling draw on each triangle, which in turn called draw on all of their triangles, until the recursion depth is reached.

What I Learned

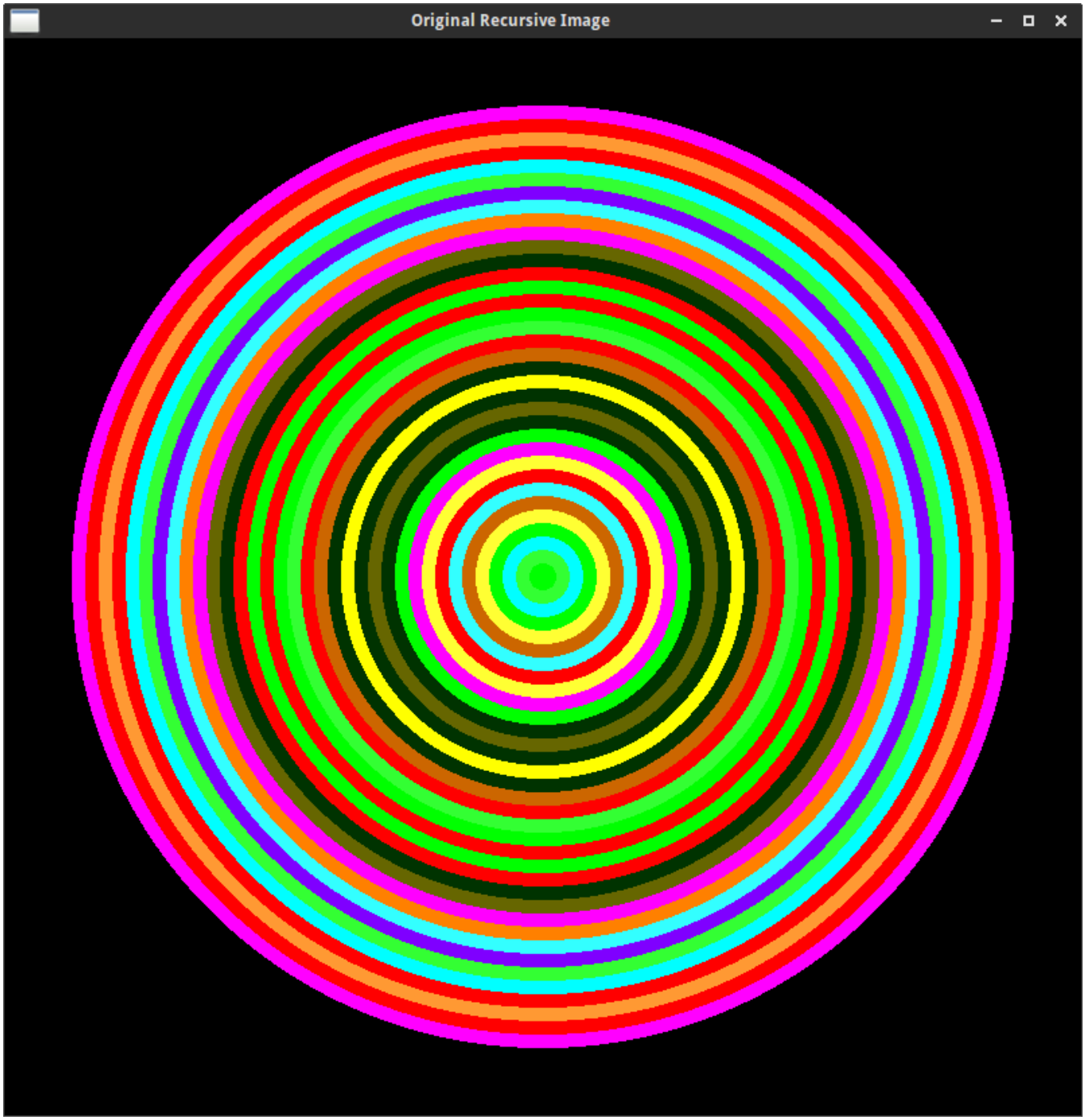
I ended up learning a little bit about recursion – I had used it before in previous classes, but never quite like this, where I was able to use pointers to other objects in order to recursively go back and draw them out. Implementing my own design after learning how to draw an object recursively was also much easier, as I already knew how I could set up the code, I just had to think about what kind of image I wanted to draw and how to make it look interesting. In the end, it was a fun assignment that taught me about recursion in a different way than I was used to.

Screenshots

Sierpinski Triangle



Original Recursive Image



Note: I also uploaded the original design to imgur a while back.
You can check out the animation at the following links:

5FPS version: https://imgur.com/e3kgJPT
60FPS version (lots of colors): https://imgur.com/qeALOGf

Source Code for PS1 Recursive Graphic

Makefile

```
1     # Makefile for the sierpinski project / original recursive image
2     # Flags to save on typing all this out
3     CC = g++
4     CFLAGS = -Wall -Werror -ansi -pedantic
5     LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
6
7     # Make both projects
8     all:  sierpinski original
9
10    # Original image related code
11    original:   o_main.o original.o
12                $(CC) o_main.o original.o -o original $(LFLAGS)
13
14    o_main.o:   o_main.cpp original.hpp
15                $(CC) -c o_main.cpp $(CFLAGS)
16
17    original.o: original.cpp original.hpp
18                $(CC) -c original.cpp $(CFLAGS)
19
20
21    # Sierpinski triangle related code
22    sierpinski: main.o sierpinski.o
23                $(CC) main.o sierpinski.o -o sierpinski $(LFLAGS)
24
25    main.o:           main.cpp sierpinski.hpp
26                $(CC) -c main.cpp $(CFLAGS)
27
28    sierpinski.o:     sierpinski.cpp sierpinski.hpp
29                $(CC) -c sierpinski.cpp $(CFLAGS)
30
31    # Cleanup
32    clean:
33                rm *.o
34                rm sierpinski
35                rm original
```

Main.cpp

```
1   /*
2    * Copyright 2015 Jason Downing
3    *
4   */
5   #include "sierpinski.hpp"
6
7   int main(int argc, char* argv[]) {
8     // Make sure we are given exactly 3 arguments
9     if ( argc < 3 || argc > 4 ) {
10        // Let the user know the correct way of calling the program.
11        std::cout << "./sierpinski [recursion-depth] [side-length] \n";
12        return -1;
13    }
14
15    int depth = atoi(argv[1]);
16    int side = atoi(argv[2]);
17
18    cout << "depth: " << depth << endl;
19    cout << "side: " << side << endl;
20
21    // Added this since I do not think it would be logical
22    // to have negative (-2, -3, etc) recursion
23    if (depth < 0) {
24        std::cout << "depth should be greater than 0 - ";
25        std::cout << "illogical to have negative recursion.\n";
26        return -2;
27    }
28
29    // Sierpinkski object, calls default constructor
30    Sierpinski obj(depth, side);
31
32    int window_height =static_cast<int>(.5)*sqrt(3.0)*static_cast<float>(side);
33
34    sf::RenderWindow window(sf::VideoMode(side, window_height), "Sierpinkski");
35
36    // Change the framerate to make it easier to see the image moving.
37    window.setFramerateLimit(1);
38
39    // Window loop
40    while (window.isOpen()) {
41        // Process events
42        sf::Event event;
43
44        while (window.pollEvent(event)) {
45          // Close window : exit
46          if (event.type == sf::Event::Closed) {
47              window.close();
48          } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
49              window.close();
50          }
51        }
52
53        window.clear();
54        window.draw(obj);   // Call the draw object in the sierpinkski class
```

```
55            window.display();
56        }
57
58        return 0;
59    }
60
```

Sierpinski.hpp

```
1   #ifndef SIERPINSKI_HPP
2   #define SIERPINSKI_HPP
3
4   #include <cmath>
5   #include <iostream>
6   #include <SFML/Graphics.hpp>
7   #include <vector>
8
9   using namespace std;
10
11  class Sierpinski : public sf::Drawable
12  {
13  public:
14    // Constructor that takes just depth and side
15    Sierpinski(int depth, int side);
16
17    // Constructor that takes three points and depth
18    Sierpinski(sf::Vector2f top, sf::Vector2f left,
                 sf::Vector2f right, int depth);
19
20    // Destructor
21    ~Sierpinski();
22
23  private:
24
25    // Draw method
26    void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;
27
28    // Member variables
29    int _depth;
30    sf::Vector2f _top, _left, _right;
31    sf::Vector2f _p1, _p2, _p3;
32
33    // Pointers to additional triangles.
34    Sierpinski *_triangle1, *_triangle2, *_triangle3;
35  };
36
37  #endif
38
```

Sierpinski.cpp

```cpp
#include <cmath>
#include <iostream>
#include <SFML/Graphics.hpp>
#include <vector>
#include "sierpinski.hpp"

using namespace std;

/*

          _top
            .
           / \
          /   \
         /     \
     p1 .-------. p3
       / \     / \
      /   \   /   \
     /     \ /     \
    .-------.-------.
  _left    p2     _right

 The above is an ASCII representation of what this code does.
 We have _top, _left, _right member variables we use to
 to draw the triangle in the middle using p1/p2/p3

 */

// Constructor                              Initialize private variables)
Sierpinski::Sierpinski(int depth, int side) : _depth (depth)
{
  // Initialize other member variables - top, left, right
  _top = sf::Vector2f(side / 2, 0);

  // Calculate the height using triangle geometry
  float height = .5 * sqrt(3.0) * (float) side;

  // Calculate left and right using the height.
  _left = sf::Vector2f(0, height);
  _right = sf::Vector2f(side-1, height);

  /* Call the constructor that accepts 3 points.
   * First find p1, p2, p3.
   * To do so, use the midpt formula:
   *
   * ( ( (x1 + x2) / 2) , ( (y1 + y2) / 2) )
   *
   */
  _p1 = sf::Vector2f( ((_top.x + _left.x) / 2), ((_top.y + _left.y) / 2));
  _p2 = sf::Vector2f( ((_left.x + _right.x) / 2), ((_left.y + _right.y) / 2));
  _p3 = sf::Vector2f( ((_top.x + _right.x) / 2), ((_top.y + _right.y) / 2));

  if (_depth - 1 > 0)
  {
```

```
55      // Now that we have the points, create 3 triangles
56      _triangle1 = new Sierpinski(_p1, _left, _p2, depth - 1);
57      _triangle2 = new Sierpinski(_top, _p1, _p3, depth - 1);
58      _triangle3 = new Sierpinski(_p3, _p2, _right, depth - 1);
59    }
60    else {
61      _triangle1 = NULL;
62      _triangle2 = NULL;
63      _triangle3 = NULL;
64    }
65  }
66
67
68  Sierpinski::Sierpinski(sf::Vector2f top, sf::Vector2f left,
69                         sf::Vector2f right, int depth) : _depth (depth)
70  {
71    // End of the recurrsion.
72    if(depth <= 0)
73    {
74      // Set the 3 triangle pointers to null
75      _triangle1 = NULL;
76      _triangle2 = NULL;
77      _triangle3 = NULL;
78
79      return;
80    }
81
82    // Set member variables - points of the triangle
83    _top = top;
84    _left = left;
85    _right = right;
86
87    // Set the filled triangle points - use midpoint formula
88    _p1 = sf::Vector2f( ((top.x + left.x) / 2), ((top.y + left.y) / 2) );
89    _p2 = sf::Vector2f( ((left.x + right.x) / 2), ((left.y + right.y) / 2) );
90    _p3 = sf::Vector2f( ((top.x + right.x) / 2), ((top.y + right.y) / 2) );
91
92    // Now that we have the points, create 3 triangles
93    _triangle1 = new Sierpinski(_p1, left, _p2, depth - 1);
94    _triangle2 = new Sierpinski(top, _p1, _p3, depth - 1);
95    _triangle3 = new Sierpinski(_p3, _p2, right, depth - 1);
96  }
97
98
99  // Destructor
100 Sierpinski::~Sierpinski()
101 {
102   // Need to destroy all the objects that were created
103   if(_triangle1 != NULL)
104   {
105     // Recusively call the destructor
106     delete _triangle1;
107     delete _triangle2;
108     delete _triangle3;
109   }
110
111   // After we clear the allocated objects, we don't have to worry about
```

```
112   // stuff allocated on the stack since that will be handled automagically.
113 }
114
115
116 void Sierpinski::draw(sf::RenderTarget& target,
117                       sf::RenderStates states) const {
118   // Check the beginning dimensions of the overall triangle
119 //   cout << "_depth is: " << _depth << endl;
120 //   cout << "_top is: (" << _top.x << ", " << _top.y << ") \n";
121 //   cout << "_left is (" << _left.x << ", " << _left.y << ") \n";
122 //   cout << "_right is (" << _right.x << ", " << _right.y << ") \n";
123
124   // The outside triangle.
125   sf::ConvexShape triangle;
126   triangle.setPointCount(3);
127   triangle.setPoint(0, _left);
128   triangle.setPoint(1, _right);
129   triangle.setPoint(2, _top);
130
131   // The filled upside down triangle.
132   sf::ConvexShape triangle2;
133   triangle2.setPointCount(3);
134   triangle2.setPoint(0, _p1);
135   triangle2.setPoint(1, _p2);
136   triangle2.setPoint(2, _p3);
137   triangle2.setFillColor(sf::Color::Red);
138
139   // Draw the two triangles above.
140   target.draw(triangle);
141
142   // Make _depth just print out a blank triangle
143   if(_depth > 0)
144   {
145     target.draw(triangle2);
146   }
147
148   if(_triangle1 != NULL)
149   {
150     _triangle1->draw(target, states);
151     _triangle2->draw(target, states);
152     _triangle3->draw(target, states);
153   }
154 }
```

o_main.cpp

```cpp
/*
 * Copyright 2015 Jason Downing
 *
*/
#include "original.hpp"

int main(int argc, char* argv[]) {
  // Make sure we are given exactly 2 arguments
  if (argc < 2 || argc > 3) {
      // Let the user know the correct way of calling the program.
      std::cout << "./original [recursion-depth]" << "\n";
      return -1;
  }

  int depth = atoi(argv[1]);

  Original obj(depth);

  sf::RenderWindow window(sf::VideoMode(800, 800),
                                        "Original Recursive Image");

  // Change the framerate to make it easier to see the image moving.
  window.setFramerateLimit(60);

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

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

      window.clear();
      window.draw(obj);     // Call the draw object in the Original class
      window.display();
  }

  return 0;
}

```

original.hpp

```
1     /*
2      * Copyright 2015 Jason Downing
3      *
4     */
5     #ifndef ORIGINAL_HPP
6     #define ORIGINAL_HPP
7
8     #include <stdlib.h>
9     #include <SFML/Graphics.hpp>
10    #include <time.h>
11    #include <cmath>
12    #include <iostream>
13    #include <vector>
14
15    class Original : public sf::Drawable {
16     public:
17      // Constructor that takes just depth and side
18      explicit Original(int depth);
19
20      // Constructor that takes three points and depth
21      Original(float radius, int depth);
22
23      // Destructor
24      ~Original();
25
26     private:
27      // Draw method
28      void virtual draw(sf::RenderTarget& target,
29                                  sf::RenderStates states) const;
30
31      // Member variables
32      int _depth;
33      float _radius;
34
35      // Pointers to additional triangles.
36      Original *_circle;
37    };
38
39    #endif
```

original.cpp

```
1     /*
2      * Copyright 2015 Jason Downing
3      *
4     */
5     #include "original.hpp"
6
7     /*
8      * Inital coordinates of the circle will be:
9      * 1) set the origin to be radius, radius - this makes it so that
10           the circle's center for transforming / moving / etc
11           is its middle point.
12     * 2) set the position to be the center of the screen
13     */
14
15
16    // Constructor with just the depth as a parameter
17    Original::Original(int depth) {
18      // Seed the rand function for random colors - uses current time to do so
19      srand(time(NULL));
20
21      // Set the inital depth
22      _depth = depth;
23
24      // Set the inital radius.
25      _radius = 350;
26
27      // Create a new circle;
28      _circle = new Original(_radius - 10, _depth - 1);
29    }
30
31
32    // Constructor with coordinates - recusive
33    Original::Original(float radius, int depth) {
34      // When depth is less than 0, the recusion is done.
35      if (depth < 0) {
36          _circle = NULL;
37
38          return;
39      }
40
41      // Set member variables
42      _depth = depth;
43      _radius = radius;
44
45      // Create a new circle;
46      _circle = new Original(_radius - 10, _depth - 1);
47      return;
48    }
49
50
51    // Destructor
52    Original::~Original() {
53      if (_circle != NULL) {
54          delete _circle;
```

```
55      }
56    }
57
58
59    // Draw method
60    void Original::draw(sf::RenderTarget& target,
61                                  sf::RenderStates states) const {
62      std::cout << "_depth is: " << _depth << std::endl;
63      std::cout << "_Radius is: " << _radius << std::endl;
64
65      // First circle
66      sf::CircleShape shape(_radius);
67      shape.setOrigin(_radius, _radius);
68      shape.setPosition(400, 400);
69      shape.setPointCount(10000);
70
71      unsigned int seed = time(NULL);
72
73      // Make a random number between 1 and 19
74      // That means there will be 19 different colors flashing!
75      int random_number = rand_r(&seed) % 19 + 1;
76
77      // Make three random numbers, trip Udit.
78
79      // Color object
80      sf::Color color(0, 0, 0, 255);
81
82      switch (random_number) {
83          case 1:
84            // Red Color
85            shape.setFillColor(sf::Color::Red);
86            break;
87
88          case 2:
89            // Orange Color
90            color.r = 255;
91            color.g = 128;
92            color.b = 0;
93            shape.setFillColor(color);
94            break;
95
96          case 3:
97            // Yellow Color
98            shape.setFillColor(sf::Color::Yellow);
99            break;
100
101         case 4:
102           // Green
103           shape.setFillColor(sf::Color::Green);
104           break;
105
106         case 5:
107           // Magenta
108           shape.setFillColor(sf::Color::Magenta);
109           break;
110
111         case 6:
```

```
112           // Cyan
113           shape.setFillColor(sf::Color::Cyan);
114           break;
115
116         case 7:
117           // Purple
118           color.r = 127;
119           color.g = 0;
120           color.b = 255;
121           shape.setFillColor(color);
122           break;
123
124         case 8:
125           // Pink
126           color.r = 255;
127           color.g = 0;
128           color.b = 255;
129           shape.setFillColor(color);
130           break;
131
132
133         case 9:
134           // Dark Orange
135           color.r = 204;
136           color.g = 102;
137           color.b = 0;
138           shape.setFillColor(color);
139           break;
140
141
142         case 10:
143           // Dark Blue
144           color.r = 0;
145           color.g = 0;
146           color.b = 102;
147           shape.setFillColor(color);
148           break;
149
150
151         case 11:
152           // Dark Purple
153           color.r = 255;
154           color.g = 0;
155           color.b = 0;
156           shape.setFillColor(color);
157           break;
158
159         case 12:
160           // Dark Green
161           color.r = 0;
162           color.g = 51;
163           color.b = 0;
164           shape.setFillColor(color);
165           break;
166
167         case 13:
168           // Gold
```

```
169           color.r = 102;
170           color.g = 102;
171           color.b = 0;
172           shape.setFillColor(color);
173           break;
174
175         case 14:
176           // Lime Green
177           color.r = 51;
178           color.g = 255;
179           color.b = 51;
180           shape.setFillColor(color);
181           break;
182
183         case 15:
184           // Lime Green
185           color.r = 51;
186           color.g = 255;
187           color.b = 51;
188           shape.setFillColor(color);
189           break;
190
191         case 16:
192           // Light Orange
193           color.r = 255;
194           color.g = 153;
195           color.b = 51;
196           shape.setFillColor(color);
197           break;
198
199         case 17:
200           // Light Yellow
201           color.r = 255;
202           color.g = 255;
203           color.b = 51;
204           shape.setFillColor(color);
205           break;
206
207         case 18:
208           // Light Blue
209           color.r = 51;
210           color.g = 255;
211           color.b = 255;
212           shape.setFillColor(color);
213           break;
214
215         case 19:
216           // Light Red
217           color.r = 255;
218           color.g = 0;
219           color.b = 0;
220           shape.setFillColor(color);
221           break;
222
223         default:
224           break;
225     }
```

```
226
227     /*
228      * Colors to use
229      * I generated these using:
230      * http://www.rapidtables.com/web/color/RGB_Color.htm
231      * I then used the Color() constructor which takes a
232      * Red, Green, Blue int for setting
233      * custom colors.
234      *
235      *  "Standard colors"
236      *  Red       -> sf::Color::Color(255, 0, 0)
237      *  Orange    -> sf::Color::Color(255, 128, 0)
238      *  Yellow    -> sf::Color::Color(255, 255, 51)
239      *  Green     -> sf::Color::Color(0, 153, 0)
240      *  Blue      -> sf::Color::Color(0, 0, 255)
241      *  Purple    -> sf::Color::Color(127, 0, 255)
242      *  Pink      -> sf::Color::Color(255, 0, 255)
243      *
244      *  "Dark colors"
245      *  Dark Orange:  sf::Color::Color(204, 102, 0)
246      *  Dark Blue:    sf::Color::Color(0, 0, 102)
247      *  Dark Purple:  sf::Color::Color(255, 0, 0)
248      *  Dark Green:   sf::Color::Color(0, 51, 0)
249      *  Gold:         sf::Color::Color(102, 102, 0)
250      *
251      *  "Light colors"
252      *  Lime Green:   sf::Color::Color(51, 255, 51)
253      *  Light Orange: sf::Color::Color(255, 153, 51)
254      *  Light Yellow: sf::Color::Color(255, 255, 51)
255      *  Light Blue:   sf::Color::Color(51, 255, 255s)
256      *  Light Blue:   sf::Color::Color(51, 153, 255)
257      *  Light Purple: sf::Color::Color(153, 51, 255)
258      *  Light Red:    sf::Color::Color(255, 0, 0)
259      *
260      */
261
262     // Draw this circle
263     target.draw(shape);
264
265     // Recusive calls
266     if (_circle != NULL) {
267         _circle->draw(target, states);
268     }
269   }
270
```

PS2a: Linear Feedback Shift Register and Unit Testing

The Assignment

This assignment required us to implement Princetom's Linear Feedback Shift Register. This type of register shift all bits left one position, and then XOR's the left most bit and the seed bit to fill the empty space on the far right side after the shift left. Our main goals were to implement the shift register in a class called "LFSR" and to implement several unit tests using the Boost test framework.

Key Concepts

We used the Boost test framework to test our LFSR class, which in my code I represent the shift register as a C++ string. I also used an integer to hold the tap position, and this allowed me to use C++'s string / ostringstream objects to implement the step and generate methods. Shifting left was accomplished by feeding the string representing the register into an ostringstream object and then appending the result of XORing the left most bit with the tap position. The Boost test framework was used to test our LFSR class, by using Boost's auto rest case method's to test the step / generate methods against edge cases, typical cases and even some of Princeton's test cases.

What I Learned

This assignment taught me a lot about testing in C++. I had never really thought much about testing my code using unit tests – in the past, I've done a combination of compiling, making sure the program runs, and then manually testing different aspects to see if it looks "OK". Using the Boost test framework made things simple – I could write tests and then change my code and with a few commands know that I did not just break code that was previously working. In the future I plan to continue using unit testing for projects I work on that get complex / hard to manually test often.

Screenshots

NEED TO RUN THIS IN LINUX.

Source Code for PS2a Linear Feedback Shift Register

Makefile

```
1     # Makefile for the sierpinski project / original recursive image
2     # Flags to save on typing all this out
3     CC= g++
4     CFLAGS= -Wall -Werror -ansi -pedantic
5     Boost= -lboost_unit_test_framework
6
7     # Make both projects
8     all:  main.out ps2a
9
10    # Boost unit tests
11    ps2a: test.o LFSR.o
12          $(CC) test.o LFSR.o -o ps2a $(Boost)
13
14    test.o:     test.cpp LFSR.hpp
15          $(CC) -c test.cpp LFSR.hpp $(CFLAGS)
16
17    # Main tester
18    main.out:   main.o LFSR.o
19          $(CC) main.o LFSR.o -o main.out
20
21    main.o:           main.cpp LFSR.hpp
22          $(CC) -c main.cpp $(CFLAGS)
23
24    LFSR.o:           LFSR.cpp LFSR.hpp
25          $(CC) -c LFSR.cpp $(CFLAGS)
26
27    # Cleanup
28    clean:
29          rm *.o
30          rm *.out
31          rm ps2a
```

main.cpp

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include "LFSR.hpp"

// Testing the LFSR class

int main()
{
  LFSR test("01101000010", 8);

  std::cout << "Test case 1. \n";

  // Test case 1 from the princeton site.

  for(int i = 0; i < 10; i++)
  {
      int bit = test.step();
      std::cout << test << " " << bit << std::endl;
  }

  LFSR test2("01101000010", 8);
  std::cout << "\n\nTest case 2.\n";

  // Test case 2 from the princeton site.
  for(int i = 0; i < 10; i++)
  {
      int r = test2.generate(5);
      std::cout << test2 << " " << r << std::endl;
  }

  LFSR test3("0101", 2);
  int num = test3.generate(2);

  std::cout << test3 << " " << num <<  std::endl;

  LFSR test4("0011010010", 3);

  int num2 = test4.generate(4);

  std::cout << num2;

  return 0;
}
```

LFSR.hpp

```
1   #ifndef LFSR_HPP
2   #define LFSR_HPP
3
4   #include <iostream>
5
6   class LFSR {
7   public:
8     LFSR(std::string seed, int t);     // Constructor
9     int step();                        // simulates one step
10    int generate(int k);               // simulates k steps
11
12    // Overloaded << operator
13    friend std::ostream& operator<< (std::ostream &out, LFSR &cLFSR);
14
15  private:
16    std::string bits;                  // holds the LFSR
17    int tap;
18  };
19
20  #endif
```

LFSR.cpp

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include "LFSR.hpp"

// Implementation file for the LFSR class

// Constructor
LFSR::LFSR(std::string seed, int t)
{
  // Sets inital seed / tap position
  bits = seed;
  tap = t;
}


// simulates one step
int LFSR::step()
{
  /*
   * We need to XOR the left most position with the tap position.
   * The left most position is just bits[0] - the very left most bit.
   *
   * The tap position can be found using the following formula:
   * tap position = size() - tap - 1
   *
   * It works by converting the tap position value into array indexes
   *
   */

  // Find the index of the tap position.
  int tap_pos = bits.length() - tap - 1;

  // XOR the tap position with the left most bit
  int res = bits[0] ^ bits[tap_pos];

  // We need to go through and shift all bits left now.
  std::string::size_type i;
  std::ostringstream ostring;   // Using string streams to append the
                                // the XOR result

  // Now shift everything left.
  for(i = 0; (unsigned)i < bits.length() - 1; i++)
  {
    ostring << bits[i + 1];
  }

  // Append the XOR result
  ostring << res;

  // Save the updated string - ostring.str() converts a string stream
  // back into a string
  bits = ostring.str();

```

```
55    // Return the XOR result bit
56    return res;
57  }
58
59
60  // simulates k steps
61  int LFSR::generate(int k)
62  {
63    // Intialize variable to zero.
64    int x = 0;
65
66    for(int i = 0; i < k; i++)
67    {
68      x = (x * 2) + step();
69    }
70
71    return x;
72  }
73
74
75  // returns a string rep. of the LFSR
76  std::ostream& operator<< (std::ostream &out, LFSR &cLFSR)
77  {
78    out << cLFSR.bits;
79    return out;
80  }
```

test.cpp

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include "LFSR.hpp"

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

// The initial test that was in this file.
BOOST_AUTO_TEST_CASE(fiveBitsTapAtTwo)
{
  LFSR l("00111", 2);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);

  LFSR l2("00111", 2);
  BOOST_REQUIRE(l2.generate(8) == 198);
}


// My first test. I just tested what Princeton's
// website gave as examples.
// In this case, they use 11 bit seeds with a tap of 8.
BOOST_AUTO_TEST_CASE(PrincetonExamples)
{
  // The simulate step test
  LFSR test("01101000010", 8);

  BOOST_REQUIRE(test.step() == 1);
  BOOST_REQUIRE(test.step() == 1);
  BOOST_REQUIRE(test.step() == 0);
  BOOST_REQUIRE(test.step() == 0);
  BOOST_REQUIRE(test.step() == 1);
  BOOST_REQUIRE(test.step() == 0);
  BOOST_REQUIRE(test.step() == 0);
  BOOST_REQUIRE(test.step() == 1);
  BOOST_REQUIRE(test.step() == 0);
  BOOST_REQUIRE(test.step() == 0);

  // The generate test from Princeton
  LFSR test2("01101000010", 8);
  BOOST_REQUIRE(test2.generate(5) == 25);
  BOOST_REQUIRE(test2.generate(5) == 4);
  BOOST_REQUIRE(test2.generate(5) == 30);
  BOOST_REQUIRE(test2.generate(5) == 27);
  BOOST_REQUIRE(test2.generate(5) == 18);
  BOOST_REQUIRE(test2.generate(5) == 26);
```

```
55    BOOST_REQUIRE(test2.generate(5) == 28);
56    BOOST_REQUIRE(test2.generate(5) == 24);
57    BOOST_REQUIRE(test2.generate(5) == 23);
58    BOOST_REQUIRE(test2.generate(5) == 29);
59  }
60
61
62  // A couple of tests making sure the constructor functions
63  // as intended. This also tests the << operator as well.
64  BOOST_AUTO_TEST_CASE(Constructor_Tests)
65  {
66    LFSR test("001100", 5);
67    std::stringstream buffer;
68    buffer << test;
69
70    // Make sure the constructor saves the seed correctly.
71    BOOST_REQUIRE(buffer.str().compare("001100") == 0);
72
73    // Try a much larger seed - 30 bits for example.
74    LFSR test2("00000001111111110010101011", 10);
75    buffer.str("");   // Clear the stringstream object
76    buffer.clear();
77    buffer << test2;
78
79    // Make sure the constructor saves the seed correctly.
80    BOOST_REQUIRE(buffer.str().compare("00000001111111110010101011") == 0);
81
82    // Now try a very small seed - 1 bit for example.
83    LFSR test3("1", 1);
84    buffer.str("");   // Clear the stringstream object
85    buffer.clear();
86    buffer << test3;
87
88    // Make sure the constructor saves the seed correctly.
89    BOOST_REQUIRE(buffer.str().compare("1") == 0);
90  }
91
92
93
```

PS2b: Image Encoding

The Assignment

This assignment builds on the previous assignment (PS2a). Using the LFSR class we built in PS2a, we were tasked with creating a program that reads in a photo from the command line and then outputs the same image, but encoded (encypted). The LFSR class was used to encode the image by left shifting all the bits in the image – thus encoding it using XOR. We also had to display the image to an SFML window and save the encypted image to a file.

Key Concepts

The main thing that this assignment used was the LFSR class from the previous homework, PS2a. The LFSR class that we built uses a shift register to store bits and has two methods, step and generate, that we used to left shift all the bits. We also used several SFML objects, such as textures, images and sprites to read in the file, encode the file and output the final encoded image to both the screen and disk. The image class was the main way we encoded the image – we were able to get both the red, green and blue pixel using .getPixel(), as seen on line #58 in the main.cpp file.

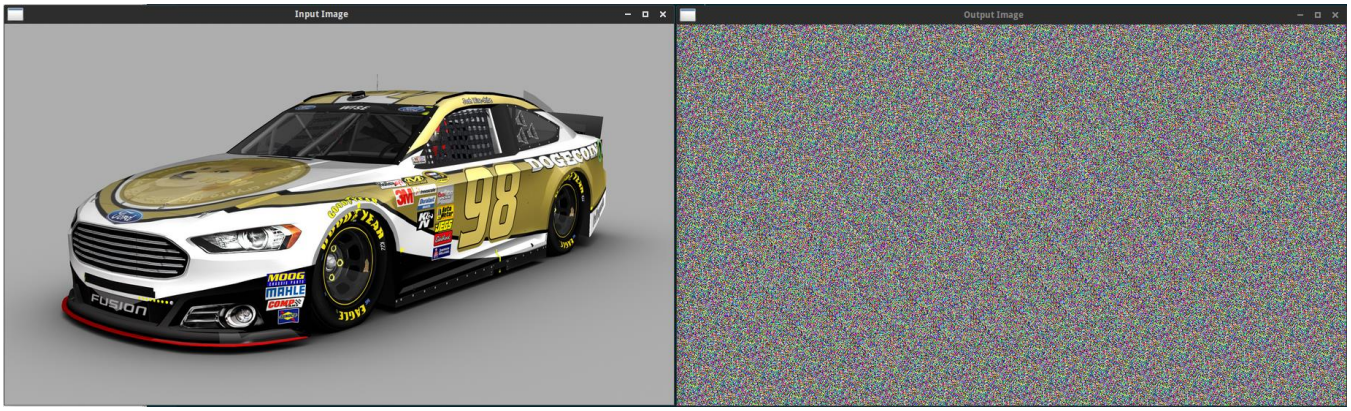
What I Learned

This project was quite fun to work with, as it used previous material that we had already built. In a way, it was neat to build something for one homework assignment and then get to reuse it to build another assignment. There was not much new material that I learned from this assignment though – since it uses the LFSR class, most of that material carries over. Perhaps most useful was the encoding portion of the assignment though. Playing around with pixels and XORing them to get an encoded image, and then displaying the final encypted image was pretty cool. However, that portion of the assignment also uses XOR, so I didn't really learn anything new per say. I mostly found other uses for the LFSR class, and that was an interesting learning experience – getting to reuse old code for a different purpose.

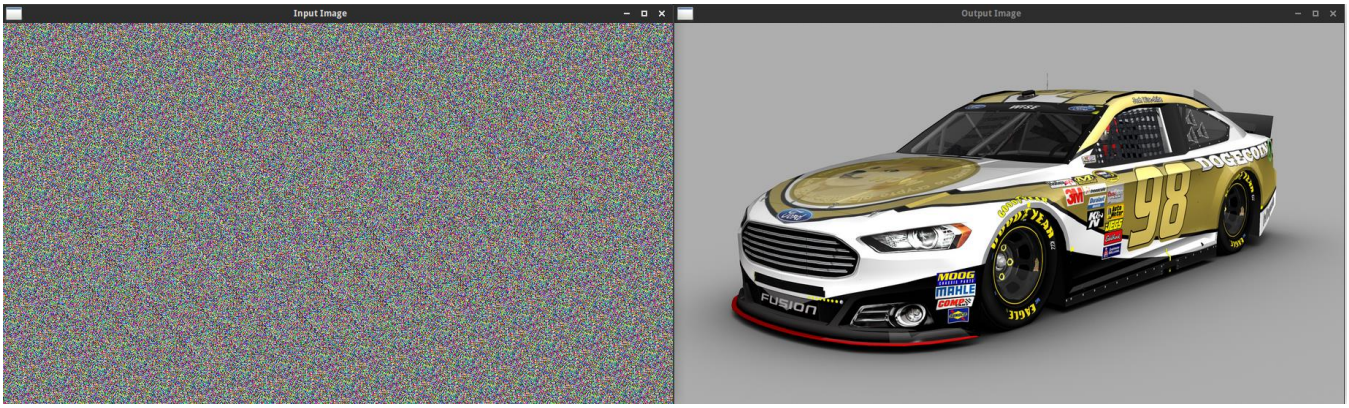
Screenshots

Original image – the #98 DOGECAR

Encoding:



Decoding:



Source Code for PS2b Image Encoding

Makefile

```
1   # Makefile for the sierpinski project / original recursive image
2   # Flags to save on typing all this out
3   CC= g++
4   CFLAGS= -Wall -Werror -ansi -pedantic
5   SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system
6
7   # Make ps2b
8   all:    PhotoMagic
9
10  # ps2b executable
11  PhotoMagic: PhotoMagic.o LFSR.o
12      $(CC) PhotoMagic.o LFSR.o -o PhotoMagic $(SFMLFLAGS)
13
14  # object files
15  PhotoMagic.o:   PhotoMagic.cpp LFSR.hpp
16      $(CC) -c PhotoMagic.cpp LFSR.hpp $(CFLAGS)
17
18  LFSR.o:     LFSR.cpp LFSR.hpp
19      $(CC) -c LFSR.cpp $(CFLAGS)
20
21  # Cleanup
22  clean:
23      rm *.o
24      rm PhotoMagic
25
```

Photomagic.cpp (main)

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "LFSR.hpp"


int main(int argc, char* argv[])
{
  // Must enter
  if(argc != 5)
  {
    std::cout << "Usage: $ ./PhotoMagic [input file]";
    std::cout << "[output file] [seed] [tap] \n";
    return -1;
  }

  // Save the command line arguments to variables
  std::string input_filename(argv[1]);
  std::string output_filename(argv[2]);
  std::string seed(argv[3]);
  int tap = atoi(argv[4]);

  // LSFR stuff
  LFSR randomizer(seed, tap);

  // Setup the two images
  sf::Image input_image;
  if (!input_image.loadFromFile(input_filename))
  {
    return -1;
  }

  sf::Image output_image;
  if (!output_image.loadFromFile(input_filename))
  {
    return -1;
  }

  // p is a pixel
  sf::Color p;

  // Setup the two windows
  sf::Vector2u size = input_image.getSize();
  sf::RenderWindow input_window(sf::VideoMode(size.x, size.y),
                                "Input Image");
  sf::RenderWindow output_window(sf::VideoMode(size.x, size.y),
                                "Output Image");

  // Randomize the bits in the image
  for(int x= 0; x < (signed)size.x; x++)
  {
```

```
55      for(int y = 0; y < (signed)size.y; y++)
56      {
57        // Get the current pixel from the input image
58        p = input_image.getPixel(x, y);
59
60        // XOR the pixels
61        p.r = p.r ^ randomizer.generate(tap);
62        p.g = p.g ^ randomizer.generate(tap);
63        p.b = p.b ^ randomizer.generate(tap);
64
65        // Modify just the output image
66        output_image.setPixel(x, y, p);
67      }
68    }
69
70    // Load the images into textures
71    sf::Texture input_texture, output_texture;
72    input_texture.loadFromImage(input_image);
73    output_texture.loadFromImage(output_image);
74
75    // Then load the textures into sprites
76    sf::Sprite input_sprite, output_sprite;
77    input_sprite.setTexture(input_texture);
78    output_sprite.setTexture(output_texture);
79
80    // Window loop
81    while (input_window.isOpen() && output_window.isOpen())
82    {
83      sf::Event event;
84
85      while (input_window.pollEvent(event))
86      {
87        if (event.type == sf::Event::Closed)
88        {
89          input_window.close();
90        }
91      }
92
93      while (output_window.pollEvent(event))
94      {
95        if (event.type == sf::Event::Closed)
96        {
97          output_window.close();
98        }
99      }
100
101     input_window.clear();
102     input_window.draw(input_sprite);    // Input image
103     input_window.display();
104
105     output_window.clear();
106     output_window.draw(output_sprite);    // Output image
107     output_window.display();
108   }
109
110   // Save the modified image to the output file
111   if (!output_image.saveToFile(output_filename))
```

```
112   {
113     return -1;
114   }
115
116   return 0;
117 }
```

LFSR.hpp

```
1   #ifndef LFSR_HPP
2   #define LFSR_HPP
3
4   #include <iostream>
5
6   class LFSR {
7   public:
8     LFSR(std::string seed, int t);     // Constructor
9     int step();                        // simulates one step
10    int generate(int k);               // simulates k steps
11
12    // Overloaded << operator
13    friend std::ostream& operator<< (std::ostream &out, LFSR &cLFSR);
14
15  private:
16    std::string bits;                  // holds the LFSR
17    int tap;
18  };
19
20  #endif
```

LFSR.cpp

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include "LFSR.hpp"

// Implementation file for the LFSR class

// Constructor
LFSR::LFSR(std::string seed, int t)
{
  // Sets inital seed / tap position
  bits = seed;
  tap = t;
}


// simulates one step
int LFSR::step()
{
  /*
   * We need to XOR the left most position with the tap position.
   * The left most position is just bits[0] - the very left most bit.
   *
   * The tap position can be found using the following formula:
   * tap position = size() - tap - 1
   *
   * It works by converting the tap position value into array indexes
   *
   */

  // Find the index of the tap position.
  int tap_pos = bits.length() - tap - 1;

  // XOR the tap position with the left most bit
  int res = bits[0] ^ bits[tap_pos];

  // We need to go through and shift all bits left now.
  std::string::size_type i;
  std::ostringstream ostring;   // Using string streams to append the
                                // the XOR result

  // Now shift everything left.
  for(i = 0; (unsigned)i < bits.length() - 1; i++)
  {
    ostring << bits[i + 1];
  }

  // Append the XOR result
  ostring << res;

  // Save the updated string - ostring.str() converts a string stream
  // back into a string
  bits = ostring.str();

  // Return the XOR result bit
```

```
56    return res;
57  }
58
59
60  // simulates k steps
61  int LFSR::generate(int k)
62  {
63    // Intialize variable to zero.
64    int x = 0;
65
66    for(int i = 0; i < k; i++)
67    {
68      x = (x * 2) + step();
69    }
70
71    return x;
72  }
73
74
75  // returns a string rep. of the LFSR
76  std::ostream& operator<< (std::ostream &out, LFSR &cLFSR)
77  {
78    out << cLFSR.bits;
79    return out;
80  }
```

PS3a: **N-Body Simulation: static universe**

The Assignment

For this assignment, we worked through Princeton's N-Body Simulation problem. It sets out to model the universe on a 2D plane, using Newton's laws of gravity to make the simulation realistic. We read in two command line arguments – total simulation time and the time step – and then displayed a static universe to the screen. The finished, moving universe was implemented in PS3b. This portion of the assignment mainly focused on reading in a file from standard I/O, and using that file's data to populate sprites (displaying the various planets) at the correct location in an SFML window.

Key Concepts

For this assignment, we used a few key C++ / Linux ideas. The first was using the < command line operator to read in a file to standard I/O. Inside the main program, I just used cin to read the file's contents – someone could, if they wanted to, type all of the planet's data in manually. To read in data easily, we also overloaded the >> operator – this way, we were able to just type:

```
cin >> c_body;
```

While not required by the assignment, I also overloaded the << operator to provide an easy way to test the program – just one line will output all the data inside the bodys object to standard I/O, like so:

```
cout << c_body;
```

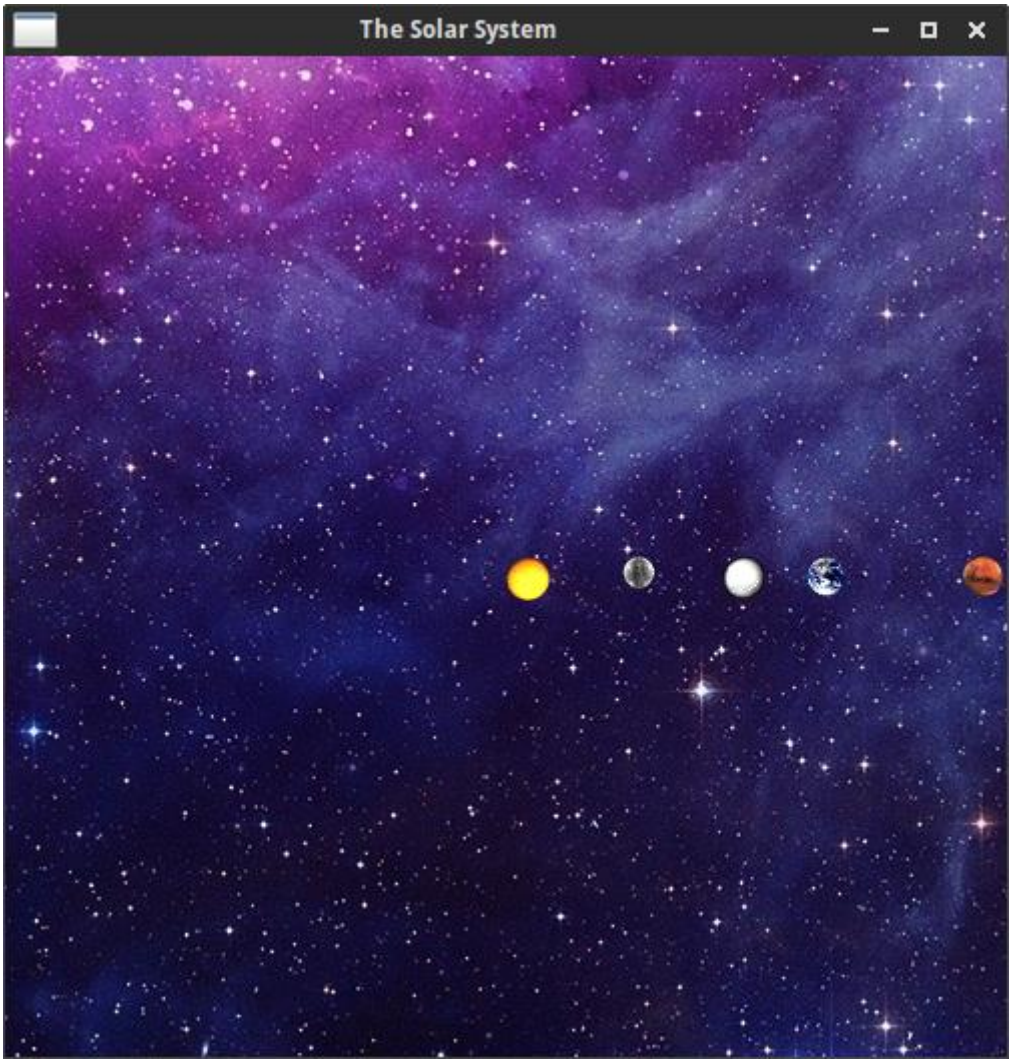
One other thing I had to do was create a method to convert X / Y positions of the planets to SFML coordinates. I was able to do this by realizing that the SFML window system sets (0, 0) as the top left corner. To convert the coordinates to the SFML system, I added half the height or side of the given window to the coordinate. There is a huge comment block on lines 52 – 82 of the bodys.cpp file explaining this math in more detail.

What I Learned

One other thing we had to do involved implementing the draw method in our bodys class – this was something I hadn't really done in a while, so it was interesting to play around with the draw method to get it to work. I did not really learn much from overloading the >> operator – I had done that a few times before, so I was familiar with how to do that. In fact I even overloaded the << operator for debugging purposes, so I was quite familiar with using those operators. Likewise, we've dealt with SFML's textures, images and sprite objects before, so displaying the planets wasn't a problem. Perhaps the biggest thing I learned was how to conver the coordinates from plantary units to SFML's units – this took some time to get right, as the planets wouldn't show up sometimes, and other times would show up in the wrong place. When I finally got it working though, it was pretty cool to have the planets all lined up nicely.

Screenshots

The Universe



Source Code for PS3b

Makefile

```
1   # Makefile for ps3b
2   # Flags to save on typing all this out
3   CC= g++
4   CFLAGS= -Wall -Werror -ansi -pedantic
5   SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system
6
7   # Make ps2b
8   all:    NBody
9
10  # body executable
11  NBody:  main.o body.o
12      $(CC) main.o body.o -o NBody $(SFMLFLAGS)
13
14  # object files
15  main.o: main.cpp body.hpp
16      $(CC) -c main.cpp body.hpp $(CFLAGS)
17
18  body.o: body.cpp body.hpp
19      $(CC) -c body.cpp body.hpp $(CFLAGS)
20
21  # Cleanup
22  clean:
23      rm *.o
24      rm NBody
25
```

main.cpp

```cpp
#include "body.hpp"

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

  // Get the first two numbers in the text file.
  // The first should be an int telling us how many planets there are.
  // The second should be a float telling us the radius of the universe.
  std::string num_planets, radius;

  // Use cin to redirect the input
  std::cin >> num_planets;
  std::cin >> radius;

  // Now we know how many planets + the radius.
  // Convert these from std::strings to int / float
  int number_planets = atoi(num_planets.c_str());
  float universe_radius = atof(radius.c_str());

  std::cout << "Num of planets: " << number_planets << std::endl;
  std::cout << "Radius: " << universe_radius << std::endl << std::endl;

  // Create a vector of body objects
  std::vector<body> body_vector;

  // Loop through, create 5 (or how ever many we're asked for)
  // body objects using the input file.
  for(int i = 0; i < number_planets; i++)
  {
    // Create a new object
    body* tmp = new body();

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

    // Set the radius and the planet positions.
    tmp->set_radius(universe_radius);
    tmp->set_position();

    // Save the object to the vector
    body_vector.push_back(*tmp);

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

  // SFML Window
  sf::RenderWindow window(sf::VideoMode(window_side, window_height),
                          "The Solar System");

  // Change the frame rate to make it easier to see the image moving.
  window.setFramerateLimit(1);

  // Background image
  sf::Image background_image;
```

```
55
56    // Background image
57    if (!background_image.loadFromFile("stars.jpg"))
58    {
59      return -1;    // Quit if the file doesn't exist.
60    }
61
62    // Load the image into a texture
63    sf::Texture background_texture;
64    background_texture.loadFromImage(background_image);
65
66    // Load the texture into a sprite
67    sf::Sprite background_sprite;
68    background_sprite.setTexture(background_texture);
69
70    // Set the position to make the background look cool
71    background_sprite.setPosition(sf::Vector2f(-700, -700));
72
73    // Window loop
74    while (window.isOpen())
75    {
76      // Process events
77      sf::Event event;
78
79      while(window.pollEvent(event))
80      {
81        // Close window : exit
82        if (event.type == sf::Event::Closed)
83        {
84          window.close();
85        }
86
87        // Pressing escape will quit the program.
88        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
89        {
90          window.close();
91        }
92      }
93
94      window.clear();
95
96      // Draws the starry background (
97      // black backgrounds are so lame for a solar system)
98      window.draw(background_sprite);
99
100     // Display the vector of objects
101     std::vector<body>::iterator it;
102
103     for(it = body_vector.begin(); it != body_vector.end(); it++)
104     {
105       window.draw(*it);
106     }
107
108     window.display();
109   }
110   return 0;
111 }
```

body.hpp

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

// Constants for the window size.
const int window_height = 500;
const int window_side = 500;

class body: public sf::Drawable
{
public:

  // Constructors
  body();
  body(double pos_x, double pos_y, double vel_x, double vel_y,
       double obj_mass, double radius, std::string file_name);

  void set_radius(float radius);
  void set_position();              // Sets the planets positions

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

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

private:

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

  // Member variables
  double _pos_x, _pos_y;
  double _vel_x, _vel_y;
  double _mass;
  double _radius;
  std::string _filename;

  // Image related objects
  sf::Image _image;
  sf::Sprite _sprite;
  sf::Texture _texture;
};
```

body.cpp

```cpp
#include "body.hpp"

// Default Constructor
body::body()
{
  // Does nothing since I call the setter methods and the >> operator.
  return;
}


// Constructor with parameters
body::body(double pos_x, double pos_y, double vel_x, double vel_y,
           double obj_mass, double radius, std::string file_name)
{
  // Set member variables
  _pos_x = pos_x;
  _pos_y = pos_y;
  _vel_x = vel_x;
  _vel_y = vel_y;
  _mass = obj_mass;
  _filename = file_name;

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

  // Load the image into a texture
  _texture.loadFromImage(_image);

  // Load the texture into a sprite
  _sprite.setTexture(_texture);

  // Set the position from the Vector2f for position
  _sprite.setPosition(sf::Vector2f(_pos_x, _pos_y));
}


// Sets the universe radius
void body::set_radius(float radius)
{
  _radius = radius;
  return;
}


// Sets the planets position
void body::set_position()
{
  /*
   * The math here probably needs some explaing:
   *
   * First thing I do is divide the pos by the radius. This gets me a nice
```

```
55      * ratio I can use. For the earth this value comes out to .5984.
56      *
57      * Next, I use this ratio and multiply it by EITHER half the width
58      * or height of the window (depends on whether its a x or y coordinate.
59      * x corresponds to wide, y corresponds with height)
60      * which gets me a position that is actually in the SFML coordinate system.
61      *
62      * Finally, now that I have the position in SFML coordinates, I adjust
63      * for the fact that the SFML window's center is half the height and side.
64      * Example would be a window of 500 by 500 has its center at 250, 250.
65      * The coordinates that we are given in the planets.txt file actually
66      * have their center at 0,0. So even though I have the SFML coordinates,
67      * I must add 250 to both the x and y coordinates to
68      * make the planets show up in the center of the screen.
69      *
70      * Example math for the earth:
71      * [pos to radius ratio] * [side OR height / 2]
72      * .5984 * 250 = 149.6
73      *
74      * The 149.6 in this case works perfectly with the SFML coordinate system.
75      * We just add 250 to get the center for SFML coordinate, which would
76      * come out to 399.6
77      *
78      * A quick note - y coordinates all have a '0' by default, so the below math
79      * actually just sets all the y coordinates to 250 - which is the center of
80      * the SFML height (0 to 500, 250 is the middle).
81      *
82      */
83     _pos_x = ( (_pos_x / _radius) * (window_side / 2) ) + (window_side / 2);
84     _pos_y = ( (_pos_y / _radius) * (window_height / 2) ) + (window_height / 2);
85
86     // Set the position from the Vector2f for position
87     _sprite.setPosition(sf::Vector2f(_pos_x, _pos_y));
88   }
89
90
91   // Drawable method
92   void body::draw(sf::RenderTarget& target, sf::RenderStates states) const
93   {
94     // Testing outputting an image.
95     target.draw(_sprite);
96   }
97
98
99   // Overridden operator >> for inputing from a file
100  std::istream& operator>> (std::istream &input, body &cBody)
101  {
102    // Read input into the object
103    input >> cBody._pos_x >> cBody._pos_y;
104    input >> cBody._vel_y >> cBody._vel_y;
105    input >> cBody._mass  >> cBody._filename;
106
107    // Now set up the images
108    // Just like the constructor
109
110    // Load the image into an image object
111    if (!cBody._image.loadFromFile(cBody._filename))
```

```
112   {
113     return input;    // Quit if the file doesn't exist.
114   }
115
116   // Load the image into a texture
117   cBody._texture.loadFromImage(cBody._image);
118
119   // Load the texture into a sprite
120   cBody._sprite.setTexture(cBody._texture);
121
122   // Set the initial position
123   cBody._sprite.setPosition(sf::Vector2f(cBody._pos_x, cBody._pos_y));
124
125   return input;
126 }
127
128
129 // Overriddden operator << for debugging
130 // Very useful for finding out why stuff doesn't work.
131 std::ostream& operator<< (std::ostream &output, body &cBody)
132 {
133   // For debugging, output all the data stored in the object.
134   output << "Filename: " << cBody._filename << std::endl;
135   output << "Pos (x): " << cBody._pos_x << std::endl;
136   output << "Pos (y): " << cBody._pos_y << std::endl;
137   output << "Vel (x): " << cBody._vel_x << std::endl;
138   output << "Vel (y): " << cBody._vel_y << std::endl;
139   output << "Mass: " << cBody._mass << std::endl << std::endl;
140
141   return output;
142 }
143
```

PS3b: N-Body Simulation:
Using Newton's laws of physics, animate the universe

The Assignment

This assignment builds on the previous one, PS3a. In the previous assignment, we created a static universe – in PS3b, we made the universe move and respond to Newton's law of universal gravitation and Newton's second law of motion.

Key Concepts

The main concepts for this assignment deal with Physics. They include:

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

We implemented these concepts in PS3b by using a few formulas, such as:

```
F = (G * M1 * M2) / R^2
R = square root (R2)
R2 = (Δx)^2 + (Δy)^2
Δx = x2 - x1
Δy = y2 - y1
```

These formulas can be found on lines 67 to 72 in the find_forcex method of bodys.cpp.

Using these formulas, we were able to simulate the movement of the planets throughout the universe. This was a tricky part of the assignment, as getting the formulas right is the key to making the universe work correctly.

What I Learned

I mainly learned a little physics in this assignment and with that how to implement different equations in a program. It was pretty tricky to get the equations correct, as I found out while programming this assignment, if you get just one equation wrong or slightly off, you can send all the planets into chaos. Once you've implemented them correctly, you'll get a nice simulation of the universe. Besides implementing the physics portion, I also learned how to play music using SFML's audio library. I did this as an extra credit portion of the assignment, but it was still pretty neat to get the theme song to *2001: A Space Odyssey* playing while the planets rotate around the Sun.

Screenshots

Run dis shitz in Linux.

ALSO – find the recording I made that went up on imgur and post it here!!

Source Code for PS3b

Makefile

```
1     # Makefile for ps3b
2     # Flags to save on typing all this out
3     CC= g++
4     CFLAGS= -Wall -Werror -std=c++0x -pedantic
5     SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
6
7     # Make ps2b
8     all:  NBody
9
10    # body executable
11    NBody:      main.o body.o
12          $(CC) main.o body.o -o NBody $(SFMLFLAGS)
13
14    # object files
15    main.o:     main.cpp body.hpp
16          $(CC) -c main.cpp body.hpp $(CFLAGS)
17
18    body.o:     body.cpp body.hpp
19          $(CC) -c body.cpp body.hpp $(CFLAGS)
20
21    # Cleanup
22    clean:
23          rm *.o
24          rm *.gch
25          rm NBody
26
```

main.cpp

```cpp
#include "body.hpp"

int main(int argc, char* argv[])
{
  if(argc != 3)   // We just want 3 arguments
  {
      // ./NBody 157788000.0 25000.0 < planets.txt
      std::cout << "Usage: ./NBody [simulation time]";
      std::cout << "[time step] < planets.txt\n";
      return -1;
  }

  // Get the simulation time / time step from the command line arguments
  std::string sim_time(argv[1]);
  std::string step_time(argv[2]);
  std::string::size_type sz;     // alias of size_t (this is for using stod)

  // Debugging
  std::cout << "Simulation time: " << sim_time << "\n";
  std::cout << "Time Step: " << step_time << "\n\n";

  // Convert these strings to doubles
  double simulation_time = 0;
  double simu_time = std::stod(sim_time, &sz);
  double time_step = std::stod(step_time, &sz);

  // Get the first two numbers in the text file.
  std::string num_planets, radius;

  // Use cin to redirect the input
  std::cin >> num_planets;
  std::cin >> radius;

  // Now we know how many planets + the radius.
  // Convert these from std::strings to int / float
  int number_planets = std::stoi(num_planets, &sz);
  double universe_radius = std::stod(radius, &sz);

  // Debugging
  std::cout << "Num of planets: " << number_planets << std::endl;
  std::cout << "Radius: " << universe_radius << std::endl << std::endl;

  // Create a vector of body objects
  std::vector<body> body_vector;

  // Loop through, create 5 (or how ever many we're asked for)
  // body objects using the input file.
  for(int i = 0; i < number_planets; i++)
  {
      // Create a new object
      body* tmp = new body();

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

```
55
56          // Set the radius and the planet positions.
57          tmp->set_radius(universe_radius);
58          tmp->set_position();
59
60          // Save the object to the vector
61          body_vector.push_back(*tmp);
62
63          // Test the object (debugging)
64          std::cout << *tmp;
65      }
66
67      // SFML Window
68      sf::RenderWindow window(sf::VideoMode(window_side, window_height),
69                              "The Solar System");
70
71      // Change the framerate to make it easier to see the image moving.
72      window.setFramerateLimit(60);
73
74      // Background image
75      sf::Image background_image;
76
77      // Background image
78      if (!background_image.loadFromFile("stars.jpg"))
79      {
80          return -1;    // Quit if the file doesn't exist.
81      }
82
83      // Declare and load a font
84      sf::Font time_font;
85      time_font.loadFromFile("arial.ttf");
86
87      // Text for displaying the current simulation time.
88      sf::Text time_text;
89
90      // Select the font
91      time_text.setFont(time_font);     // font is a sf::Font
92
93      // Set the character size
94      time_text.setCharacterSize(14);   // in pixels, not points!
95
96      // Set the color
97      time_text.setColor(sf::Color::White);
98
99      // Load the music file
100     sf::Music music;
101     if(!music.openFromFile("2001.ogg"))
102     {
103         return -1;    // error
104     }
105
106     // PLAY THE EPIC TUNE
107     music.play();
108
109     // Load the image into a texture
110     sf::Texture background_texture;
111     background_texture.loadFromImage(background_image);
```

```
112
113     // Load the texture into a sprite
114     sf::Sprite background_sprite;
115     background_sprite.setTexture(background_texture);
116
117     // Set the position to make the background look cool
118     // background_sprite.setPosition(sf::Vector2f(-700, -700));
119
120     std::vector<body>::iterator it;
121     std::vector<body>::iterator x, y;
122
123     while (window.isOpen())
124     {
125         sf::Event event;
126
127         while(window.pollEvent(event))
128         {
129           if (event.type == sf::Event::Closed)
130           {
131               window.close();
132           }
133         }
134
135         window.clear();
136
137         // Draws the starry background
138         // (black backgrounds are so lame for a solar system)
139         window.draw(background_sprite);
140
141         // I cast to an int to keep the time sane looking
142         time_text.setString("Elapsed time:" + std::to_string(simulation_time));
143
144         // Display the time in the left hand corner of the window
145         window.draw(time_text);
146
147         // Calculate the net force on each body object
148         x = body_vector.begin();
149         double force_x, force_y;
150
151         // First loop goes through the whole body vector
152         //    so we make sure each body object gets its net force updated.
153         for(int a = 0; a < number_planets; a++)
154         {
155           y = body_vector.begin();
156           force_x = 0;
157           force_y = 0;
158
159           // Second loop goes through the body vector again, so that
160           // the current body object gets effected by every other body object.
161           for(int b = 0; b < number_planets; b++)
162           {
163               if(a != b) // Make sure not include the force on the body itself.
164               {          // Basically - (earth, earth) shouldn't be a case.
165                 force_x += find_forcex(*x, *y);
166                 force_y += find_forcey(*x, *y);
167               }
168               y++;
```

```
169           }
170           // Update the forces inside the current object
171           x->set_forces(force_x, force_y);
172           x++;
173         }
174 
175         // Display the vector of objects
176         for(it = body_vector.begin(); it != body_vector.end(); it++)
177         {
178           window.draw(*it);
179           std::cout << *it << std::endl;
180 
181           // While we're displaying the objects, might as
182           // well move it forward one step!
183           it->step(time_step);
184           it->set_position();   // Update image position.
185         }
186 
187         window.display();
188 
189         // Increase simulation time variable by the simulation step
190         simulation_time += time_step;
191 
192         // Stop when we've reached the simulation time
193         if(simulation_time == simu_time)
194         {
195           break;
196         }
197     }
198 
199     // For debugging to compare against Princeton's data
200     // (This basically prints out the final positions / velocities / etc)
201     std::cout << "\n\n\n";
202     for(it = body_vector.begin(); it != body_vector.end(); it++)
203     {
204         std::cout << *it << std::endl;
205     }
206 
207     return 0;
```

body.hpp

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

// Constants for the window size.
const int window_height = 1000;
const int window_side = 1000;

// Physics Constants
const double gravity = 6.67e-11;

class body: public sf::Drawable
{
public:

  // Constructors
  body();
  body(double pos_x, double pos_y, double vel_x, double vel_y,
         double obj_mass, double radius, std::string file_name);

  // Set radius / image position
  void set_radius(float radius);
  void set_position();              // Sets the planets positions

  // Force related methods
  friend double find_forcex(body &Body1, body &Body2);
  friend double find_forcey(body &Body1, body &Body2);
  void set_forces(double forcex, double forcey);

  // Time step
  void step(double time_t);

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

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

private:

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

  // Member variables
  double _acc_x, _acc_y;
  double _for_x, _for_y;
  double _pos_x, _pos_y;
  double _vel_x, _vel_y;
```

```
55      double _mass, _radius;
56      std::string _filename;
57
58      // Image related objects
59      sf::Image _image;
60      sf::Sprite _sprite;
61      sf::Texture _texture;
62    };
```

body.cpp

```cpp
#include "body.hpp"

// Default Constructor
body::body()
{
  // Does nothing since I call the setter methods and the >> operator.
  return;
}


// Constructor with parameters
body::body(double pos_x, double pos_y, double vel_x, double vel_y,
                double obj_mass, double radius, std::string file_name)
{
  // Set member variables
  _pos_x = pos_x;
  _pos_y = pos_y;
  _vel_x = vel_x;
  _vel_y = vel_y;
  _mass = obj_mass;
  _filename = file_name;

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

  // Load the image into a texture
  _texture.loadFromImage(_image);

  // Load the texture into a sprite
  _sprite.setTexture(_texture);

  // Set the position from the Vector2f for position
  _sprite.setPosition(sf::Vector2f(_pos_x, _pos_y));
}


// Sets the universe radius
void body::set_radius(float radius)
{
  _radius = radius;
  return;
}


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

```

```
55
56    // Finds the force (x) between two body objects
57    double find_forcex(body &Body1, body &Body2)
58    {
59      /*
60       * Formulas:
61       * F = (G * M1 * M2) / R^2
62       * R =
63       * R^2 = R squared
64       * Δx = x2 - x1
65       * Δy = y2 - y1
66       */
67      double dx = Body2._pos_x - Body1._pos_x;
68      double dy = Body2._pos_y - Body1._pos_y;
69      double R2 = pow(dx, 2) + pow(dy, 2);
70      double R = sqrt(R2);
71      double force = (gravity * Body1._mass * Body2._mass) / R2;
72      double for_x = force * (dx / R);
73
74      std::cout << "Body1 Filename: " << Body1._filename << "\n";
75      std::cout << "dx: " << dx << "\n";
76      std::cout << "dy: " << dy << "\n";
77      std::cout << "Force: " << force << "\n";
78      std::cout << "Force(x) " << for_x << "\n";
79      std::cout << "Gravity: " << gravity << "\n";
80      std::cout << "Body1 Mass: " << Body1._mass << "\n";
81      std::cout << "Body2 Mass: " << Body2._mass << "\n\n";
82
83      return for_x;
84    }
85
86
87    // Finds the force (y) between two body objects
88    double find_forcey(body &Body1, body &Body2)
89    {
90      /*
91       * Formula is: F = (G * M1 * M2) / R^2
92       */
93      double dx = Body2._pos_x - Body1._pos_x;
94      double dy = Body2._pos_y - Body1._pos_y;
95      double R2 = pow(dx, 2) + pow(dy, 2);
96      double R = sqrt(R2);
97      double force = (gravity * Body1._mass * Body2._mass) / R2;
98      double for_y = force * (dy / R);
99
100     std::cout << "Body1 Filename: " << Body1._filename << "\n";
101     std::cout << "dx: " << dx << "\n";
102     std::cout << "dy: " << dy << "\n";
103     std::cout << "Force: " << force << "\n";
104     std::cout << "Force(y " << for_y << "\n";
105     std::cout << "Gravity: " << gravity << "\n";
106     std::cout << "Body1 Mass: " << Body1._mass << "\n";
107     std::cout << "Body2 Mass: " << Body2._mass << "\n\n";
108
109     return for_y;
110   }
111
```

```
112
113   void body::step(double time_t)
114   {
115     /*
116      * Convert forces into acceleration
117      *
118      *  F =  m * a
119      * Ax = Fx / m
120      * Ay = Fy / m
121      *
122      */
123     _acc_x = _for_x / _mass;
124     _acc_y = _for_y / _mass;
125
126     /*
127      * Calculate change in velocity
128      *
129      * dvelx = (ax * time_step)
130      * _velx = = _velx + (ax * time_step)
131      *
132      * (vx + Δt ax, vy + Δt ay)
133      *
134      */
135     _vel_x = _vel_x + (_acc_x * time_t);
136     _vel_y = _vel_y + (_acc_y * time_t);
137
138     /*
139      * Body moves based on its velocity
140      *
141      * _xpos = _xpos + (_xvel * time_step)
142      *
143      * (px + Δt vx, py + Δt vy)
144      *
145      */
146     _pos_x = _pos_x + (_vel_x * time_t);
147     _pos_y = _pos_y + (_vel_y * time_t);
148
149   }
150
151
152   // Sets the planets position
153   void body::set_position()
154   {
155     /*
156      * The math here probably needs some explaining:
157      *
158      * First thing I do is divide the pos by the radius. This gets me a nice
159      * ratio I can use. For the earth this value comes out to .5984.
160      *
161      * Next, I use this ratio and multiply it by EITHER half the width or
162      * height of the window (depends on whether its a x or y coordinate.
163      * x corresponds to wide, y corresponds with height)
164      * which gets me a position that is actually in the SFML coordinate system.
165      *
166      * Finally, now that I have the position in SFML coordinates, I adjust
167      * for the fact that the SFML window's center is half the height and side.
168      * Example would be a window of 500 by 500 has its center at 250, 250.
```

```
169      * The coordinates that we are given in the planets.txt file actually
170      * have their center at 0,0. So even though I have the SFML coordinates,
171      * I must add 250 to both the x and y coordinates to
172      * make the planets show up in the center of the screen.
173      *
174      * Example math for the earth:
175      * [pos to radius ratio] * [side OR height / 2]
176      * .5984 * 250 = 149.6
177      *
178      * The 149.6 in this case works perfectly with the SFML coordinate system.
179      * We just add 250 to get the center for SFML coordinate, which would come
180      * out to 399.6
181      *
182      * A quick note: y coordinates all have a '0' by default, so the below math
183      * actually just sets all the y coordinates to 250 - which is the center of
184      * the SFML height (0 to 500, 250 is the middle).
185      *
186      */
187
188     // Note - using temp variables to avoid modifying the member variables and
189     // having to switch back/forth between astronomical and pixel measurements.
190     double pos_x = ((_pos_x / _radius) * (window_side/2)) + (window_side/2);
191     double pos_y = ((_pos_y/_radius) * (window_height/2)) + (window_height/2);
192
193     // Set the position from the Vector2f for position
194     // Flip the x and y positions for going counter-clockwise
195     _sprite.setPosition(sf::Vector2f(pos_y, pos_x));
196   }
197
198
199   // Drawable method
200   void body::draw(sf::RenderTarget& target, sf::RenderStates states) const
201   {
202     // Testing outputting an image.
203     target.draw(_sprite);
204   }
205
206
207   // Overridden operator >> for inputing from a file
208   std::istream& operator>> (std::istream &input, body &cBody)
209   {
210     // Read input into the object
211     input >> cBody._pos_x >> cBody._pos_y;
212     input >> cBody._vel_x >> cBody._vel_y;
213     input >> cBody._mass  >> cBody._filename;
214
215     // Now set up the images
216     // Just like the constructor
217
218     // Load the image into an image object
219     if (!cBody._image.loadFromFile(cBody._filename))
220     {
221         return input;    // Quit if the file doesn't exist.
222     }
223
224     // Load the image into a texture
225     cBody._texture.loadFromImage(cBody._image);
```

```
226
227     // Load the texture into a sprite
228     cBody._sprite.setTexture(cBody._texture);
229
230     // Set the initial position
231     cBody._sprite.setPosition(sf::Vector2f(cBody._pos_x, cBody._pos_y));
232
233     // Set force / acceleration to 0.
234     cBody._for_x = 0;
235     cBody._for_y = 0;
236     cBody._acc_x = 0;
237     cBody._acc_y = 0;
238
239     return input;
240   }
241
242
243   // Overriddden operator << for debugging
244   // Very useful for finding out why stuff doesn't work.
245   std::ostream& operator<< (std::ostream &output, body &cBody)
246   {
247     // For debugging, output all the data stored in the object.
248     output << "Filename: " << cBody._filename << std::endl;
249     output << "Acceleration (x): " << cBody._acc_x << std::endl;
250     output << "Acceleration (y): " << cBody._acc_y << std::endl;
251     output << "Force (x): " << cBody._for_x << std::endl;
252     output << "Force (y): " << cBody._for_y << std::endl;
253     output << "Pos (x): " << cBody._pos_x << std::endl;
254     output << "Pos (y): " << cBody._pos_y << std::endl;
255     output << "Vel (x): " << cBody._vel_x << std::endl;
256     output << "Vel (y): " << cBody._vel_y << std::endl;
257     output << "Mass: " << cBody._mass << std::endl;
258     output << "Radius: " << cBody._radius << std::endl << std::endl;
259
260     return output;
261   }
```

PS4: Edit Distance

The Assignment

For PS4, we implemented a program to find the optimal alignment of two strings. Princeton calls it the alignment of two DNA strings. A key idea for this program was also to use dynamic programming to make calculating the edit distance efficient.

Key Concepts

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

We also were able to recover the path that our algortithm took by retracing our steps through the matrix. We did this by using a few rules:

1. The optimal alignment matches `x[i]` up with `y[j]`. In this case, we must have `opt[i][j]` = `opt[i+1][j+1]` if `x[i]` equals `y[j]`, or `opt[i][j]` = `opt[i+1][j+1] + 1` otherwise.
2. The optimal alignment matches `x[i]` up with a gap. In this case, we must have `opt[i][j]` = `opt[i+1][j] + 2`.
3. The optimal alignment matches `y[j]` up with a gap. In this case, we must have `opt[i][j]` = `opt[i][j+1] + 2`.

Source: Princeton's webpage for this assignment.

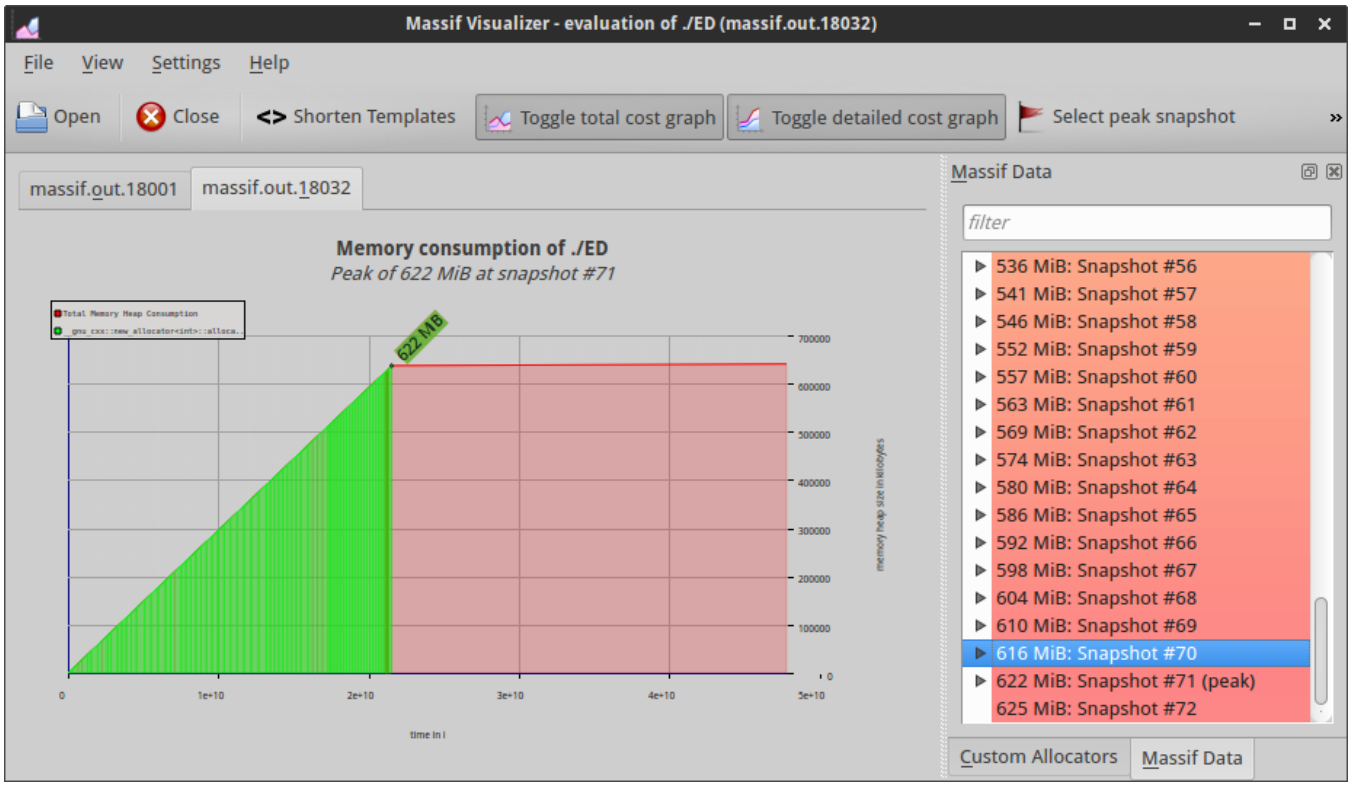
Using these three rules, we will able to go from the top left most cell of the matrix ([0][0]), where we originally found the final edit distance, and then trace our steps back to the bottom right most cell ([N][M]).

What I Learned

I learned a few things from this assignment. First, that vectors are pretty inefficient compared to c arrays – seeing the results of other classmates' implementations on iSENSE showed me that. I also got to play around with valgrind, which I've used in the past. I found a nice way of visualizing valgrind's results as well using massif visualizer. See the screenshot from it in the screenshots section. Also, using the Needleman-Wunsch method to calculate subproblems was pretty interesting – that was not something I had really considered before, so it has given me an insight into other methods of programming.

Screenshots

LINUX…



Screenshot of Massif Visualizer, showing how much RAM PS4 used on my machine.

Source Code for PS4

Makefile

```
1   # Makefile for the sierpinski project / original recursive image
2   # Flags to save on typing all this out
3   CC= g++
4   CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
5   SFLAGS= -lsfml-system
6
7   # Make ps4
8   all:    ED
9
10  # body executable
11  ED: main.o ED.o
12      $(CC) main.o ED.o -o ED $(SFLAGS)
13
14  # object files
15  main.o: main.cpp ED.hpp
16      $(CC) -c main.cpp ED.hpp $(CFLAGS)
17
18  ED.o:   ED.cpp ED.hpp
19      $(CC) -c ED.cpp ED.hpp $(CFLAGS)
20
21  # Cleanup
22  clean:
23      rm *.o
24      rm *.gch
25      rm ED
```

main.cpp

```cpp
#include "ED.hpp"

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

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

  // Check stdin
//   std::cout << "String 1: " << string1 << "\n";
//   std::cout << "String 2: " << string2 << "\n";

  // Declare a ED object
  ED ed_test(string1, string2);

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

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

  // Print out the edit distance
  std::cout << "Edit distance = " << distance << "\n";
  std::cout << alignment; // this should have newlines inside of it!

  // Debug the Matrix
  ed_test.PrintMatrix();

  // I'm printing this out again to make it easier
  // when running tests for the README.
  std::cout << "\nEdit distance = " << distance;
  t = clock.getElapsedTime();
  std::cout << "\nExecution time is " << t.asSeconds() << " seconds \n";

  return 0;
}
```

ED.hpp

```
1   #ifndef ED_HPP
2   #define ED_HPP
3
4   #include <iostream>
5   #include <iomanip>
6   #include <sstream>
7   #include <string>
8   #include <stdexcept>      // std::out_of_range
9   #include <vector>
10  #include <SFML/System.hpp>
11
12  class ED
13  {
14    public:
15      ED();
16      ED(std::string string_one, std::string string_two);
17      ~ED();
18      int penalty(char a, char b);
19      int min(int a, int b, int c);
20      int OptDistance();
21      std::string Alignment();
22
23      // Debug method - just prints out what the Matrix should look like.
24      void PrintMatrix();
25
26    private:
27      std::string _string_one, _string_two;
28
29      // Large vector, which holds inside of it vectors of type integer.
30      std::vector< std::vector<int> > _matrix;
31  };
32
33  #endif
```

ED.cpp

```cpp
#include "ED.hpp"

// Default constructor
ED::ED()
{

}


// Contructor with parameters
ED::ED(std::string string_one, std::string string_two)
{
  _string_one = string_one;
  _string_two = string_two;
}


// Destructor
ED::~ED()
{

}


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

  else if(a != b)
  { // not equal and no spaces
      return 1;
  }

  // If something fails, return a -1.
  // We can check this for errors.
  return -1;
}


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

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

```
55      }
56
57      else if(c < a && c < b)
58      {
59          return c;
60      }
61
62      // They are all equal if we get here.
63      // So just return a cause w/e
64      return a;
65    }
66
67
68    // Finds the optimal distance between the two strings
69    int ED::OptDistance()
70    {
71      // This is where we should do some sort of loop and populate
72      // the matrix. We should use the penalty and min methods as well.
73
74      int i, j;
75      int N = _string_one.length();
76      int M = _string_two.length();
77
78      // Create the Matrix
79      /*
80       * Quick note - it seems the vector of a vector of ints actually runs
81       * like this:
82       * ------------------> this is one vector
83       * ------------------> another vector
84       * ------------------> etc
85       *
86       * So each column is a vector of ints.
87       * And the entire vector is a vector of a vector of ints.
88       * With each vector inside the large vector pointing to vectors,
89       * which are technically the columns.
90       *
91       * This is important for some of the math down below, which follows
92       * Princeton's site but can get confusing due to how the vectors seem
93       * to be making themselves.
94       *
95       */
96      for(i = 0; i <= M; i++)
97      {
98          std::vector<int> tmp;
99          _matrix.push_back(tmp);
100
101         // Now push '0's back into the given vector
102         for(j = 0; j <= N; j++)
103         {
104           _matrix.at(i).push_back(0);
105         }
106     }
107
108     // Start by filling out the bottom row
109     for(i = 0; i <= M; i++)
110     {
111         // Very bottom row
```

```
112         _matrix[i][N] = 2 * (M - i);
113     }
114
115     // Now fill out the side row.
116     for(j = 0; j <= N; j++)
117     {
118         // Very right most column
119         _matrix[M][j] = 2 * (N - j);
120     }
121
122     // Now that we have the initial sides, we can go ahead and
123     // calculate the rest of the sub problems.
124     for(i = M - 1; i >= 0; i--)
125     {
126         for(j = N - 1; j >= 0; j--)
127         {
128           // Using Princeton's formula, we can just calculate every single row!
129           /*
130            * Note - min of 3 numbers, which we must get
131            *      from using the penalty method.
132            */
133           int opt1 = _matrix[i+1][j+1] +
                          penalty(_string_one[j], _string_two[i]);
134           int opt2 = _matrix[i+1][j] + 2;
135           int opt3 = _matrix[i][j+1] + 2;
136
137           _matrix[i][j] = min(opt1, opt2, opt3);
138         }
139     }
140
141     return _matrix[0][0];
142   }
143
144
145   // This is a test method which will print out the matrix, so we can compare
146   // against Princeton's site to see if we're doing it right.
147   void ED::PrintMatrix()
148   {
149     std::cout << "\n\nPrinting out the Matrix for debug purposes: \n\n";
150
151     // Iterator the large vector
152     std::vector< std::vector<int> >::iterator a;
153
154     // Iterator to each vector of integers.
155     std::vector<int>::iterator b;
156
157     for(a = _matrix.begin(); a != _matrix.end(); a++)
158     {
159         for(b = (*a).begin(); b != (*a).end(); b++)
160         {
161           // Using std::right and setw(3) to align numbers to the right.
162           // See the stackoverflow post in the README for an example
163           std::cout << std::right << std::setw(3) << *b << " ";
164         }
165         std::cout << "\n";
166     }
167   }
```

```
168
169
170   // Returns the alignment
171   // Here we should trace the matrix and find the string
172   // that displays the actual alignment.
173   std::string ED::Alignment()
174   {
175   // Let's declare a stringstream object to hold the string we want to return.
176     std::ostringstream return_string;
177
178     // Get M & N for going through the Matrix
179     // NOTE: I use standard MxN Matrix notation - that is,
180     // M is the number of rows, N is the number of columns.
181     int M = _string_two.length();
182     int N = _string_one.length();
183
184     int i = 0, j = 0;
185     int pen, opt1, opt2, opt3;
186     std::string ret_str;
187
188     // More debug stuff to check where the code is heading.
189   //   std::cout << "We want: " << _matrix[M-1][N-1] << "\n";
190
191     // A while loop will work here since we want to move either diagonally,
192     // down or right.
193     while(i < M || j < N) // Need to run until we hit the far bottom right!
194     {
195         // Checking vector bounds with try/catch
196         try{
197           pen =  penalty(_string_one[j], _string_two[i]);
198           opt1 = _matrix.at(i+1).at(j+1) + pen;
199         }
200         catch(const std::out_of_range& error)
201         {
202           // out of range
203           opt1 = -1;
204         }
205         try{
206           opt2 = _matrix.at(i+1).at(j) + 2;
207         }catch(const std::out_of_range& error)
208         {
209           // out of range
210           opt2 = -1;
211         }
212         try{
213           opt3 = _matrix.at(i).at(j+1) + 2;
214         }catch(const std::out_of_range& error)
215         {
216           // out of range
217           opt3 = -1;
218         }
219
220         // Move diagonally
221         if(_matrix[i][j] == opt1)
222         {
223           return_string << _string_one[j] << " ";
224           return_string <<  _string_two[i] << " "  << pen << "\n";
```

```
225           i++;
226           j++;
227         }
228
229         // Move down
230         else if(_matrix[i][j] == opt2)
231         {
232           return_string << "- " << _string_two[i] << " 2\n";
233           i++;
234         }
235
236         // Move right
237         else if(_matrix[i][j] == opt3)
238         {
239           return_string << _string_one[j] << " -" << " 2\n";
240           j++;
241         }
242     }
243
244     // Get the string from the ostringstream object & return it.
245     ret_str = return_string.str();
246     return ret_str;
247   }
```

PS5a: Ring Buffer

The Assignment

For this assignment, we implemented a RingBuffer, complete with unit tests and exceptions. Implementing the RingBuffer was the main goal of the assignment, and it works by wrapping around like a circle array in order to store values – in our case, 16 bit integers. Boost unit tests were also used to check the RingBuffer for errors, and the RingBuffer was designed to throw certain exceptions for specific errors. Some exceptions include std::invalid argument if you attempt to create a RingBuffer with a capacity of 0 or less, and std::runtime_error if you attempt to enqueue a full RingBuffer or dequeue or peek at an empty RingBuffer.

Key Concepts

The main concepts for this assignment focused on implementing the RingBuffer using exceptions, which is something we had not really discussed in class before this assignment. Try / Catch blocks were used to test for invalid actions. Another key idea was using Google's cpplint python script to check our code for consistency. This was introduced in the optional PSX assignment, but it was the first time we began to use cpplint in class. Boost was also used to check the RingBuffer, and ensure that the buffer threw the correct exceptions when it was supposed to, and no exceptions were thrown for valid actions.

What I Learned

Exceptions and cpplint were what I really took away from this assignment. Having only used exceptions a few times in Computing III, I was somewhat rusty at them. It did not take too long to figure them out though, and having used them in this class, I was able to apply exceptions to another class – Android Development, which uses Java and Java, like C++, uses exceptions a fair amount. Cpplint was also an interesting tool to use. Mainly in the class absolutely hate it, but I found it to be useful at times. It forced me to change some bad coding habbits, such as using really long lines (80+ characters in length). At the same time though, it forced me to program Google's way, which is annoying if you've already got a programming style that you like (which I did somewhat have). For example, I used to like putting brackets on a separate line, but after using cpplint I've become used to putting them on the same line as the if / loop statement. This was hard to get used to, but now that I am used to it I've pretty much stuck with that style for other programming assignments that I do outside of class.

Screenshots

RUN THIS IN LINUX

Source Code for PS5a

Makefile

```
1   # Makefile for PS5a
2   # Flags to save on typing all this out
3   CC= g++
4   CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
5   Boost= -lboost_unit_test_framework
6
7   # Make ps5a & a dummy tester
8   all:    ps5a main.out
9
10  # PS5 executable
11  ps5a:   test.o RingBuffer.o
12      $(CC) test.o RingBuffer.o -o ps5a $(Boost)
13
14  # main.out is just a basic test executable
15  main.out: main.o RingBuffer.o
16      $(CC) main.o RingBuffer.o -o main.out
17
18  # Object files
19  RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
20      $(CC) -c RingBuffer.cpp RingBuffer.hpp $(CFLAGS)
21
22  test.o: test.cpp RingBuffer.hpp
23      $(CC) -c test.cpp RingBuffer.hpp $(CFLAGS)
24
25  main.o: main.cpp RingBuffer.hpp
26      $(CC) -c main.cpp RingBuffer.hpp $(CFLAGS)
27
28  # Cleanup object files
29  clean:
30      rm *.o
31      rm *.gch
32      rm ps5a
33      rm *.out
```

main.cpp

```cpp
/*
 * Copyright 2015 Jason Downing
 * All rights reserved.
 * MIT Licensed - see http://opensource.org/licenses/MIT for details.
 *
 */
#include "RingBuffer.hpp"

int main() {
  std::cout << "Test main.\n";

  RingBuffer test(100);
  test.enqueue(1);
  test.enqueue(2);
  test.enqueue(3);
  std::cout << "Peek: " << test.peek() << "\n";

  std::cout << "Deq 1: " << test.dequeue() << "\n";
  std::cout << "Deq 2: " << test.dequeue() << "\n";


  test.output();

  // Test looping back around
  RingBuffer test2(3);

  test2.enqueue(1);
  test2.enqueue(2);
  test2.enqueue(3);

  test2.dequeue();
  test2.dequeue();
  test2.dequeue();

  test2.enqueue(1);
  test2.enqueue(2);
  test2.enqueue(3);
  test2.dequeue();
  test2.enqueue(4);

  test2.output();

  return 0;
}

```

RingBuffer.hpp

```
1   /*
2    * Copyright 2015 Jason Downing
3    * All rights reserved.
4    * MIT Licensed - see http://opensource.org/licenses/MIT for details.
5    *
6    */
7   #include <stdint.h>
8   #include <iostream>
9   #include <string>
10  #include <sstream>
11  #include <exception>
12  #include <stdexcept>
13  #include <vector>
14
15  class RingBuffer {
16   public:
17    // API functions
18
19    // Empty ring buffer, with given max capacity.
20    explicit RingBuffer(int capacity);
21    int size();                   // return # of items in the buffer.
22    bool isEmpty();               // is size == 0?
23    bool isFull();                // is size == capacity?
24    void enqueue(int16_t x);      // add item x to the end.
25    int16_t dequeue();            // delete and return item from the front
26    int16_t peek();               // return (don't delete) item from the front.
27
28    // Other functions
29    void output();
30
31   private:
32    std::vector<int16_t> _buffer;
33    int _first;
34    int _last;
35    int _capacity;
36    int _size;
37  };
38
```

RingBuffer.cpp

```
1  /*
2   * Copyright 2015 Jason Downing
3   * All rights reserved.
4   * MIT Licensed - see http://opensource.org/licenses/MIT for details.
5   *
6   */
7  #include "RingBuffer.hpp"
8
9  // Create an empty ring buffer, with given max capacity.
10 RingBuffer::RingBuffer(int capacity) {
11   if (capacity < 1) {
12     throw
13     std::invalid_argument("RB constructor: capacity must be greater than zero");
14   }
15
16   _last = 0;
17   _first = 0;
18   _size = 0;
19   _capacity = capacity;
20   _buffer.resize(capacity);
21
22   return;
23 }
24
25
26 // Return # of items in the buffer.
27 int RingBuffer::size() {
28   return _size;
29 }
30
31
32 // Is size == 0?
33 bool RingBuffer::isEmpty() {
34   // Determine if the RingBuffer is empty.
35   if (_size == 0) {
36     return true;
37   } else {
38     return false;
39   }
40 }
41
42
43 // Is size == capacity?
44 bool RingBuffer::isFull() {
45   // Determine if size equals capacity.
46   if (_size == _capacity) {
47     return true;
48   } else {
49     return false;
50   }
51 }
52
53
54 // Add item x to the end.
```

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

```
112// Dumps the variables to stdout
113void RingBuffer::output() {
114  std::cout << "   First: " << _first << "\n";
115  std::cout << "    Last: " << _last << "\n";
116  std::cout << "Capacity: " << _capacity << "\n";
117  std::cout << "    Size: " << _size << "\n";
118  std::cout << "Vector size: " << _buffer.size() << "\n";
119  std::cout << "Vector capacity: " << _buffer.capacity() << "\n";
120  std::cout << "Buffer (no blanks): \n";
121
122  int x = 0;
123  int y = _first;
124
125  while (x < _size) {
126    // Make the loop go back to 0 to continue printing.
127    if (y >= _capacity) {
128      y = 0;
129    }
130
131    std::cout << _buffer[y] << " ";
132    y++;
133    x++;
134  }
135
136  std::cout << "\nDump the entire buffer (including blanks): \n";
137
138  for (int x = 0; x < _capacity; x++) {
139    std::cout << _buffer[x] << " ";
140  }
141
142  std::cout << "\n\n";
143}
144
```

test.cpp

```
1     /*
2      * Copyright 2015 Jason Downing
3      * All rights reserved.
4      * MIT Licensed - see http://opensource.org/licenses/MIT for details.
5      *
6      */
7     #define BOOST_TEST_DYN_LINK
8     #define BOOST_TEST_MODULE Main
9     #include <boost/test/unit_test.hpp>
10
11    #include "RingBuffer.hpp"
12
13    // Tests various aspects of the contructor.
14    BOOST_AUTO_TEST_CASE(Constructor) {
15      // Normal constructor - shouldn't fail.
16      BOOST_REQUIRE_NO_THROW(RingBuffer(100));
17
18      // These should fail.
19      BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
20      BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
21      BOOST_REQUIRE_THROW(RingBuffer(-1), std::invalid_argument);
22    }
23
24
25    // Checks the size() method
26    BOOST_AUTO_TEST_CASE(Size) {
27      RingBuffer test(1);
28
29      // This should be size 0.
30      BOOST_REQUIRE(test.size() == 0);
31
32      test.enqueue(5);
33
34      // This should be size 1.
35      BOOST_REQUIRE(test.size() == 1);
36
37      test.dequeue();
38      BOOST_REQUIRE(test.size() == 0);
39    }
40
41
42    // Checks the isEmpty() method
43    BOOST_AUTO_TEST_CASE(isEmpty) {
44      // This should be true
45      RingBuffer test(5);
46      BOOST_REQUIRE(test.isEmpty() == true);
47
48      // This should be false
49      RingBuffer test2(5);
50      test2.enqueue(5);
51      BOOST_REQUIRE(test2.isEmpty() == false);
52    }
53
54
```

```
55    // Checks the isFull() method
56    BOOST_AUTO_TEST_CASE(isFull) {
57      RingBuffer test(5);
58      BOOST_REQUIRE(test.isFull() == false);
59
60      RingBuffer test2(1);
61      test2.enqueue(5);
62      BOOST_REQUIRE(test2.isFull() == true);
63    }
64
65
66    // Test enqueue
67    BOOST_AUTO_TEST_CASE(Enqueue) {
68      // These test basic enqueuing
69      RingBuffer test(5);
70
71      BOOST_REQUIRE_NO_THROW(test.enqueue(1));
72      BOOST_REQUIRE_NO_THROW(test.enqueue(2));
73      BOOST_REQUIRE_NO_THROW(test.enqueue(3));
74      BOOST_REQUIRE_NO_THROW(test.enqueue(4));
75      BOOST_REQUIRE_NO_THROW(test.enqueue(5));
76      BOOST_REQUIRE_THROW(test.enqueue(6), std::runtime_error);
77    }
78
79
80    // Test dequeue
81    BOOST_AUTO_TEST_CASE(Dequeue) {
82      RingBuffer test(5);
83
84      test.enqueue(0);
85      test.enqueue(1);
86      test.enqueue(2);
87
88      BOOST_REQUIRE(test.dequeue() == 0);
89      BOOST_REQUIRE(test.dequeue() == 1);
90      BOOST_REQUIRE(test.dequeue() == 2);
91      BOOST_REQUIRE_THROW(test.dequeue(), std::runtime_error);
92    }
```

PS5b: Karplus-Strong String Simulation and Guitar Hero

The Assignment

In the second part of PS5, we used the RingBuffer from PS5a to create a model of a Guitar, by implementing the Karplus-Strong algorithm to simulate the plucking of a guitar string. We were tasked with implementing a few methods to simulate the playing of a guitar – such as pluck, tic, sample, etc. Finally, we also made the main program, GuitarHero, respond to keyboard presses, which each key generated a different note.

Key Concepts

The main algorithm was used in this assignment was the Karplus-Strong algorithm, which was used to simulate the plucking of a guitar. The Karplus-Strong algorithm works by modeling frequencies, and it takes the first two values, averages them and then multiplies the result by the energy decay factor, which in our case was .996. This, along with the RingBuffer, allowed us to model sound (to some degree) and made it seem like a guitar string was being plucked.

What I Learned

Learning about the Karplus-Strong algorithm was quite interesting, and it gave me insight on how to model sound in a program. It was fairly tricky to get the Karplus-Strong update correct as well, and I also got a lot of segfaults when I first wrote the program using pointers. After I switched to a member initialization list I found the segfaults went away and the program was able to work somewhat. I also ran into issues with getting the sound to play, but I'm not entirely sure how I fixed that – I guess playing around with the keyboard settings for SFML helped. Which speaking of that, SFML's Keyboard library is pretty useful for controlling a piano / guitar / etc – I also thought about its uses in simple 2D games, which along with the planet stuff from PS3 could make a pretty cool space invaders like game. Perhaps one day I'll try playing around with those again for fun.

Screenshots

NEED TO MAKE SCREENSHOT IN LINUX

Source Code for PS5b

Makefile

```
1   # Makefile for PS5b
2   # Flags to save on typing all this out
3   CC= g++
4   CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
5   SFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
6   Boost= -lboost_unit_test_framework
7
8   # Make ps5a & a dummy tester
9   all:    GuitarHero GStest
10
11  # PS5B executable
12  GuitarHero: GuitarHero.o GuitarString.o RingBuffer.o
13      $(CC) GuitarHero.o GuitarString.o RingBuffer.o -o GuitarHero $(SFLAGS)
14
15  # GStest executable
16  GStest: GStest.o GuitarString.o RingBuffer.o
17      $(CC) GStest.o GuitarString.o RingBuffer.o -o GStest $(Boost)
18
19  # Object files
20  GuitarHero.o:   GuitarHero.cpp GuitarString.hpp
21      $(CC) -c GuitarHero.cpp GuitarString.hpp $(CFLAGS)
22
23  GuitarString.o: GuitarString.cpp GuitarString.hpp
24      $(CC) -c GuitarString.cpp GuitarString.hpp $(CFLAGS)
25
26  RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
27      $(CC) -c RingBuffer.cpp RingBuffer.hpp $(CFLAGS)
28
29  GStest.o:   GStest.cpp
30      $(CC) -c GStest.cpp $(Boost)
31
32  # Cleanup object files
33  clean:
34      rm *.o
35      rm *.gch
36      rm GuitarHero
37      rm GStest
```

GuitarHero.cpp (main)

```
/*
 * Copyright 2015 Jason Downing
 * All rights reserved.
 * MIT Licensed - see http://opensource.org/licenses/MIT for details.
 *
 */
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>

#include <math.h>
#include <limits.h>

#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include <vector>

#include "RingBuffer.hpp"
#include "GuitarString.hpp"

#define CONCERT_A 440.0
#define SAMPLES_PER_SEC 44100
const int keyboard_size = 37;

std::vector<sf::Int16> makeSamplesFromString(GuitarString gs) {
  std::vector<sf::Int16> samples;

  gs.pluck();
  int duration = 8;  // seconds
  int i;
  for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
    gs.tic();
    samples.push_back(gs.sample());
  }

  return samples;
}

int main() {
  sf::RenderWindow window(sf::VideoMode(800, 800), "SFML Guitar Hero");
  sf::Event event;

  // sample vector / freq - all reused.
  double freq;
  std::vector<sf::Int16> sample;

  // Samples vector of a vector
  std::vector<std::vector<sf::Int16>> samples(keyboard_size);
  std::vector<sf::SoundBuffer> buffers(keyboard_size);
  std::vector<sf::Sound> sounds(keyboard_size);

```

```
55    // From Princeton.
56    std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
57
58    // For loop to go through the entire keyboard.
59    for (int i = 0; i < (signed)keyboard.size(); i++) {
60      // Set the frequency and make a GuitarString.
61      freq = CONCERT_A * pow(2, ( (i - 24)/12.0));
62      GuitarString tmp = GuitarString(freq);
63
64      // Make a vector of sf::Int16, will act as a sample.
65      sample = makeSamplesFromString(tmp);
66      samples[i] = sample;
67
68      // Load the above same into the buffer vector.
69      if (!buffers[i].loadFromSamples(&samples[i][0],
70          samples[i].size(), 2, SAMPLES_PER_SEC)) {
71        throw
72     std::runtime_error("sf::SoundBuffer: failed to load from samples.");
73      }
74
75      // Now load the buffer into a sf::sound.
76      sounds[i].setBuffer(buffers[i]);
77    }
78
79    while (window.isOpen()) {
80      while (window.pollEvent(event)) {
81          if (event.type == sf::Event::TextEntered) {
82            // This detects unicode characters
83            if (event.text.unicode < 128) {
84              // Convert the key from unicode to ASCII
85              char key = static_cast<char>(event.text.unicode);
86
87              // Now go through the keyboard string and if we find a match,
88              // then access that sound and play it.
89              for (int i = 0; i < (signed)keyboard.size(); i++) {
90                if (keyboard[i] == key) {
91                  std::cout << "Keyboard key is: " << keyboard[i] << "\n";
92                  std::cout << "Attempting to play sound...\n";
93                  sounds[i].play();
94                  break;
95                }
96              }
97            }
98          }
99        }
100
101     window.clear();
102     window.display();
103   }
104   return 0;
105 }
106
```

GuitarString.hpp

```
1   /*
2    * Copyright 2015 Jason Downing
3    * All rights reserved.
4    * MIT Licensed - see http://opensource.org/licenses/MIT for details.
5    *
6    */
7   #ifndef GUITARSTRING_HPP
8   #define GUITARSTRING_HPP
9
10  #include <SFML/Audio.hpp>
11  #include <SFML/Graphics.hpp>
12  #include <SFML/System.hpp>
13  #include <SFML/Window.hpp>
14  #include <cmath>
15  #include <iostream>
16  #include <string>
17  #include <vector>
18  #include "RingBuffer.hpp"
19
20  const int SAMPLING_RATE = 44100;
21  const double ENERGY_DECAY_FACTOR = 0.996;
22
23  class GuitarString {
24   public:
25    // create a guitar string of the given freq using a rate of 44,100
26    explicit GuitarString(double frequency);
27
28    // create a guitar string with size and initial values of the vector init
29    explicit GuitarString(std::vector<sf::Int16> init);
30
31    // pluck the guitar string by replacing the buffer with random valuess
32    void pluck();
33    void tic();              // advance the simulation one time step
34    sf::Int16 sample();      // return the current sample
35    int time();              // return number of times tic was called so far
36   private:
37    RingBuffer _buff;
38    int _N;
39    int _tic;
40  };
41  #endif
```

GuitarString.cpp

```cpp
/*
 * Copyright 2015 Jason Downing
 * All rights reserved.
 * MIT Licensed - see http://opensource.org/licenses/MIT for details.
 *
 */
#include "GuitarString.hpp"
#include <vector>


// Create a guitar string of the given freq using a rate of 44,100
GuitarString::GuitarString(double frequency):
                                        _buff(ceil(SAMPLING_RATE / frequency)) {
  // Ringbuffer will be this large.
  _N = ceil(SAMPLING_RATE / frequency);

  // Enqueue N (44,100 / freq) 0's.
  for (int i = 0; i < _N; i++) {
    _buff.enqueue((int16_t)0);
  }

  // Set _tic to 0 for the tic / time methods.
  _tic = 0;
}


// Create a guitar string with size and initial values of the vector init
GuitarString::GuitarString(std::vector<sf::Int16> init):
                                                    _buff(init.size()) {
  // RingBuffer will be as large as the array.
  _N = init.size();

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

  // Enqueue all the items in the vector.
  for (it = init.begin(); it < init.end(); it++) {
    _buff.enqueue((int16_t)*it);
  }

  // Set _tic to 0 for the tic / time methods.
  _tic = 0;
}


// pluck the guitar string by replacing the buffer with random valuess
void GuitarString::pluck() {
  // Remove N items
  for (int i = 0; i < _N; i++) {
    _buff.dequeue();
  }

  // Add N random items between -32768 to 32767
  for (int i = 0; i < _N; i++) {
```

```
55      _buff.enqueue((sf::Int16)(rand() & 0xffff)); //NOLINT
56    }
57
58    return;
59  }
60
61
62  // advance the simulation one time step
63  void GuitarString::tic() {
64    // First get the first value, and dequeue it at the same time.
65    int16_t first = _buff.dequeue();
66
67    // Get the second value (DON'T dequeue it)
68    int16_t second = _buff.peek();
69
70    // Now we can apply the Karplus-Strong update:
71    // Take the first two values, average them and multiply by the
72    // ENERGY_DECAY_FACTOR
73    int16_t avg = (first + second) / 2;
74    int16_t karplus = avg * ENERGY_DECAY_FACTOR;
75
76    // Debugging code.
77  //   std::cout << "Karplus is: " << karplus << "\n";
78
79    // Now enqueue the Karplus-Strong update.
80    _buff.enqueue((sf::Int16)karplus);
81
82    _tic++;
83
84    return;
85  }
86
87
88  // return the current sample
89  sf::Int16 GuitarString::sample() {
90    // Get the value of the item at the front of the RingBuffer
91    sf::Int16 sample = (sf::Int16)_buff.peek();
92
93    return sample;
94  }
95
96
97  // return number of times tic was called so far
98  int GuitarString::time() {
99    return _tic;
100 }
```

RingBuffer.hpp

```
1   /*
2    * Copyright 2015 Jason Downing
3    * All rights reserved.
4    * MIT Licensed - see http://opensource.org/licenses/MIT for details.
5    *
6    */
7   #include <stdint.h>
8   #include <iostream>
9   #include <string>
10  #include <sstream>
11  #include <exception>
12  #include <stdexcept>
13  #include <vector>
14
15  class RingBuffer {
16   public:
17    // API functions
18
19    // Empty ring buffer, with given max capacity.
20    explicit RingBuffer(int capacity);
21    int size();                   // return # of items in the buffer.
22    bool isEmpty();               // is size == 0?
23    bool isFull();                // is size == capacity?
24    void enqueue(int16_t x);      // add item x to the end.
25    int16_t dequeue();            // delete and return item from the front
26    int16_t peek();               // return (don't delete) item from the front.
27
28    // Other functions
29    void output();
30
31   private:
32    std::vector<int16_t> _buffer;
33    int _first;
34    int _last;
35    int _capacity;
36    int _size;
37  };
38
```

RingBuffer.cpp

```
1  /*
2   * Copyright 2015 Jason Downing
3   * All rights reserved.
4   * MIT Licensed - see http://opensource.org/licenses/MIT for details.
5   *
6   */
7  #include "RingBuffer.hpp"
8
9  // Create an empty ring buffer, with given max capacity.
10 RingBuffer::RingBuffer(int capacity) {
11   if (capacity < 1) {
12     throw
13     std::invalid_argument("RB constructor: capacity must be greater than zero");
14   }
15
16   _last = 0;
17   _first = 0;
18   _size = 0;
19   _capacity = capacity;
20   _buffer.resize(capacity);
21
22   return;
23 }
24
25
26 // Return # of items in the buffer.
27 int RingBuffer::size() {
28   return _size;
29 }
30
31
32 // Is size == 0?
33 bool RingBuffer::isEmpty() {
34   // Determine if the RingBuffer is empty.
35   if (_size == 0) {
36     return true;
37   } else {
38     return false;
39   }
40 }
41
42
43 // Is size == capacity?
44 bool RingBuffer::isFull() {
45   // Determine if size equals capacity.
46   if (_size == _capacity) {
47     return true;
48   } else {
49     return false;
50   }
51 }
52
53
54 // Add item x to the end.
```

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

```
112// Dumps the variables to stdout
113void RingBuffer::output() {
114  std::cout << "   First: " << _first << "\n";
115  std::cout << "    Last: " << _last << "\n";
116  std::cout << "Capacity: " << _capacity << "\n";
117  std::cout << "    Size: " << _size << "\n";
118  std::cout << "Vector size: " << _buffer.size() << "\n";
119  std::cout << "Vector capacity: " << _buffer.capacity() << "\n";
120  std::cout << "Buffer (no blanks): \n";
121
122  int x = 0;
123  int y = _first;
124
125  while (x < _size) {
126    // Make the loop go back to 0 to continue printing.
127    if (y >= _capacity) {
128      y = 0;
129    }
130
131    std::cout << _buffer[y] << " ";
132    y++;
133    x++;
134  }
135
136  std::cout << "\nDump the entire buffer (including blanks): \n";
137
138  for (int x = 0; x < _capacity; x++) {
139    std::cout << _buffer[x] << " ";
140  }
141
142  std::cout << "\n\n";
143}
144
```

PS6: Markov Model of Natural Language

The Assignment

For PS6, we created a class that implements Markov chains to model English text. Markov chains are statistical model of text, which count the occurences and sequences of characters in an English word / sentence. The MarkovModel class uses several methods, such as freq() which returns the frequency of a character in a k-gram (fixed number of text), and randk() which returns a random character that follows a given k-gram. Using this MarkovModel class allows us to generate psudo-random text through the use of statistics – and specifically using the gen() method, which returns a string of random characters following a given k-gram.

Key Concepts

The key idea behind this assignment is to understand and implement Markov chains, which are statistical models of English text. To build this Markov Model class, I used a C++ map, which I set up as a std::string, int map – that is, the k-gram is the key to the map, and the int is the value, or number of occurences of the given k-gram. We had several methods that were used to model the Markov chains:

1. Freq() – two versions of this method, one for finding the frequency of k-grams and another for finding the frequency of a character following the given k-gram.
2. Randk() – generates a random character that follows the given k-gram
3. Gen() – generates a string of length T characters which simulates a trajectory through the given Markov chain.

These four methods were critical to getting the TextGenerator program working correctly. The program was designed to take an input text, and output pseudo-random text using the MarkovModel class. I was able to get my TextGenerator program to work rather well. Some example outputs of my TextGenerator program can be found in the screenshots section.

What I Learned

I learned a fair amount about Markov chains, and how to model them. I also found out that many text prediction programs use this idea, which was pretty interesting to hear about. For example, most autocorrect functions on smartphones use a Markov chain to predict what you are trying to say. They do this by determining the frequencies of characters, just like we did, and then they use these frequencies to make an educated guess on what word you are trying to type / say / etc. It was pretty neat to figure out how these programs actually work, and to implement a design that is used in the real world was pretty cool. Inputting text into the finished program is also quite fun, as you get completely unexpected outputs, but these outputs actually make sense when you think about how the text is generated – it's random, but at the same time it is based on how the original text was structured.

Screenshots

NEED TO MAKE SCREENSHOT IN LINUX.

Source Code for PS6

Makefile

```
1   # Makefile for PS6
2   # Flags to save on typing all this out
3   CC = g++
4   CFLAGS = -g -Wall -Werror -std=c++0x -pedantic
5   SFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
6   Boost = -lboost_unit_test_framework
7
8   # Make ps5a & a dummy tester
9   all:    TextGenerator mmtest
10
11  # PS6 executable
12  TextGenerator:  TextGenerator.o MarkovModel.o
13      $(CC) TextGenerator.o MarkovModel.o -o TextGenerator
14
15  mmtest: mmtest.o MarkovModel.o
16      $(CC) mmtest.o MarkovModel.o -o mmtest $(Boost)
17
18  # Object files
19  TextGenerator.o:TextGenerator.cpp MarkovModel.hpp
20      $(CC) -c TextGenerator.cpp MarkovModel.hpp $(CFLAGS)
21
22  MarkovModel.o:MarkovModel.cpp MarkovModel.hpp
23      $(CC) -c MarkovModel.cpp  MarkovModel.hpp $(CFLAGS)
24
25  mmtest.o:mmtest.cpp
26      $(CC) -c mmtest.cpp $(Boost)
27
28  # Cleanup object files
29  clean:
30      rm *.o
31      rm *.gch
32      rm TextGenerator
33      rm mmtest
```

TextGenerator.cpp

```cpp
/*
 * Copyright 2015 Jason Downing
 * All rights reserved.
 * MIT Licensed - see http://opensource.org/licenses/MIT for details.
 *
 */
#include <string>
#include "MarkovModel.hpp"

int main(int argc, const char* argv[]) {
  // Make sure the user knows how to use our amazing TextGenerator program.
  if (argc != 3) {
    std::cout << "Usage: ./TextGenerator (int K) (int T)\n";
    return 0;
  }

  // For some weird reason, need to use a string to please -fpermissive
  // Can't just cast to int with getting:
  // error: cast from ‘const char*’ to ‘int’ loses precision [-fpermissive]
  std::string str_k(argv[1]);
  std::string str_t(argv[2]);

  // Get the command line arguments as ints.
  int k = std::stoi(str_k);
  int t = std::stoi(str_t);

  // Check the arguments for santity
//   std::cout << "k is: " << k << "\n";
//   std::cout << "t is: " << t << "\n";

  // Now take input from standard IO.
  // Note: using Roy's solution from the discussion group.
  std::string input = "";
  std::string current_txt = "";   // Set these to NULL just to be sure.

  // This will read the entire file that you pipe into stdio.
  while (std::cin >> current_txt) {
    input += " " + current_txt;
    current_txt = "";
  }

  // I figured we should output the user's input for sanity checking.
  std::cout << "ORIGINAL INPUT TEXT BELOW THIS LINE.\n\n";

  // Only show the first T characters that the user cares about though.
  for (int a = 0; a < t; a++) {
    std::cout << input[a];

    // This is for formatting, I just wanted to be able to read the text.
    if (input[a] == '.' || input[a] == '!') {
      std::cout << "\n";
    }
  }

```

```
55    // Whenever stand IO hits a newline, we've finished taking input and
56    // can actually do fun text generating stuff!
57
58    // First make a final output string to use.
59    std::string output_string = "";
60
61    // We also need to use a MarkovModel object! Give it the int k as the
62    // order. Also, the input text will be our string that we pass to the
63    // constructor.
64    MarkovModel amazing(input, k);
65
66    output_string += "" + amazing.gen(input.substr(0, k), t);
67
68    // We're done! Just output the text to standard IO for the user to see. ^_^
69    std::cout << "\n\nFINAL OUTPUT TEXT BELOW THIS LINE.\n\n";
70
71    // Rather then just dump to cout, I figured I'd format this nicely too.
72    // I just added a bunch of newlines.
73    for (int a = 0; a < t; a++) {
74      std::cout << output_string[a];
75
76      // This is for formatting, I just wanted to be able to read the text.
77      if (output_string[a] == '.' || output_string[a] == '!') {
78        std::cout << "\n";
79      }
80    }
81
82    std::cout << "\n";
83
84    // Dump the object to test it.
85  //   std::cout << "\n\n" << amazing << "\n";
86
87    return 0;
88  }
```

MarkovModel.hpp

```
1   /* Copyright 2015 Jason Downing
2    * All rights reserved.
3    * MIT Licensed - see http://opensource.org/licenses/MIT for details.
4    */
5   #ifndef MARKOVMODEL_HPP
6   #define MARKOVMODEL_HPP
7   #include <algorithm>
8   #include <iostream>
9   #include <map>
10  #include <string>
11  #include <stdexcept>
12
13
14  class MarkovModel {
15   public:
16    /* Creates a Markov model of order k from the given text.
17     * Assume that the text has a length of at least k.                */
18    MarkovModel(std::string text, int k);
19
20    // Order k of Markov model
21    int order();
22
23    /* Number of occurrences of kgram in text.
24     * -> throw an exception if kgram is not of length k               */
25    int freq(std::string kgram);
26
27    /* Number of times that character c follows kgram
28     * If order = 0, return number of times char c appears
29     * -> throw an exception if kgram is not of length k               */
30    int freq(std::string kgram, char c);
31
32    /* Random character following given kgram
33     * -> Throw an exception if kgram is not of length k.
34     * -> Throw an exception if no such kgram.                         */
35    char randk(std::string kgram);
36
37    /* Generate a string of length T characters by simulating a
38     * trajectory through the corresponding Markov chain.
39     * The first k characters of the newly generated string should be
40     * the argument kgram. ** Assume that T is at least k. **
41     * -> Throw an exception if kgram is not of length k.              */
42    std::string gen(std::string kgram, int T);
43
44    /* overload the stream insertion operator and display the internal
45     * state of the Markov Model. Print out the order, the alphabet,
46     * and the frequencies of the k-grams and k+1-grams.               */
47    friend std::ostream& operator<< (std::ostream &out, MarkovModel &mm);
48
49   private:
50    int _order;
51    std::map <std::string, int> _kgrams;
52    std::string _alphabet;
53  };
54  #endif
```

MarkovModel.cpp

```
1   /* Copyright 2015 Jason Downing
2    * All rights reserved.
3    * MIT Licensed - see http://opensource.org/licenses/MIT for details.
4    */
5   #include "MarkovModel.hpp"
6   #include <algorithm>
7   #include <map>
8   #include <string>
9   #include <stdexcept>
10  #include <vector>
11  #include <utility>
12
13  /* Creates a Markov model of order k from the given text.
14   * Assume that the text has a length of at least k.                */
15  MarkovModel::MarkovModel(std::string text, int k) {
16    // Set the order.
17    _order = k;
18
19    // Seed the random number generator for later.
20    srand((int)time(NULL)); //NOLINT
21
22    // Need to treat the text as circular!
23    // So wrap around the first k characters.
24    // Add the wrap around portion.
25    std::string text_circular = text;   // Make a copy.
26
27    for (int a = 0; a < _order; a++) {
28      text_circular.push_back(text[a]);
29    }
30
31    int text_len = text.length();  // Find the text's length, easier later on.
32
33    // Now we need to set the alphabet.
34    char tmp;
35    bool inAlpha = false;
36
37    // Go through the entire text and pick out all the individual letters,
38    for (int i = 0; i < text_len; i++) {
39      tmp = text.at(i);
40      inAlpha = false;
41
42      // See if this letter has been added to the alphabet.
43      for (unsigned int y = 0; y < _alphabet.length(); y++) {
44        // tmp is already in the alphabet!
45        // Also ignore upper case,
46        if (_alphabet.at(y) == tmp) {
47          inAlpha = true;   // Match it as being in the alphabet.
48        }
49      }
50
51      // If tmp isn't in the alphabet, isAlpha should be FALSE.
52      // So push it back to the alphabet.
53      if (!inAlpha) {
54        _alphabet.push_back(tmp);
```

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

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

```
169   if (kgram.length() != (unsigned)_order) {
170     throw std::runtime_error("Error - kgram not of length k (randk)");
171   }
172
173   // Need an iterator for going through the kgrams map.
174   std::map<std::string, int>::iterator it;
175
176   // Search through and see if we find the given kgram.
177   it = _kgrams.find(kgram);
178
179   // We didn't find it. Throw an exception.
180   if (it == _kgrams.end()) {
181     throw std::runtime_error("Error: Could not find the given kgram! (randk)");
182   }
183
184   // Get the freq of the given kgram. (we will rand by this number!)
185   int kgram_freq = freq(kgram);
186
187   // Picking a random int from the possible characters.
188   // This should be an int from 1 to the total number of possible letters.
189   int random_value = rand() % kgram_freq; //NOLINT
190
191   double test_freq = 0;
192   double random_num = static_cast<double>(random_value) / kgram_freq;
193   double last_values = 0;
194
195   // Go through all the letters.
196   for (unsigned int a = 0; a < _alphabet.length(); a++) {
197     // This line basically calculates the probability as a double.
198     test_freq =  static_cast<double>(freq(kgram, _alphabet[a])) / kgram_freq;
199
200     // NOTE: I'm comparing our random number, which we got from rand()ing
201     // the kgram_freq against the test freq, and making that test_freq is
202     // NOT 0. My logic is if a letter has "0" appearences in the kgram,
203     // then its probability of showing up is exactly 0 / kgram_freq, which
204     // is 0. I add the sum of all the last values as well as that is how
205     // you do this sort of math.
206     // Credit to the amazing stackoverflow post which gave me this idea,
207     // but I actually turned it into a more "dynamic" implemenation by adding
208     // the sum of all the last frequencies.
209     if (random_num < test_freq + last_values && test_freq != 0) {
210       // Return this letter since it matches.
211       return _alphabet[a];
212     }
213
214     // I think the above comment details this nicely. Its basically a sum
215     // counter for all of the last few frequencies. I add them up here to
216     // avoid missing the second frequency by mistake. I think it would also
217     // possibly cause the below "return '-'" statement too, since if you sum
218     // something before you expect to its going to cause an error.
219     last_values += test_freq;
220   }
221
222   // This is here for error checking. We should never reach this point unless
223   // something in the above code is wrong.
224   return '-';
225 }
```

```
226
227
228 /* Generate a string of length T characters by simulating a trajectory
229  * through the corresponding Markov chain.
230  * The first k characters of the newly generated string should be
231  * the argument kgram. ** Assume that T is at least k. **
232  * -> Throw an exception if kgram is not of length k.             */
233 std::string MarkovModel::gen(std::string kgram, int T) {
234   // Throw an exception if kgram is not of length k.
235   if (kgram.length() != (unsigned)_order) {
236     throw std::runtime_error("Error - kgram not of length k. (gen)");
237   }
238
239   // We need to take the given kgram, and add "T" characters to it, based
240   // on the given kgram's frequencies and whatever we add to its frequencies.
241   // This confused me at first and I think I now understand how to deal with
242   // this function.
243
244   // We first call randk on the original kgram, then we append add it to
245   // a new string with both the original kgram and the new character.
246   // We note that "T" is the final length of the string, so if we have a
247   // order 2 kgram, and T equals 4, we gotta add just two characters.
248   // So we keep running until we've hit string length of T.
249
250   // The final string we will return. We'll build it up over time.
251   std::string final_string = "";
252
253   // Temp char for using to collect the return value from randk()
254   char return_char;
255
256   // Add the kgram to it.
257   final_string += "" + kgram;
258
259   // Now the magic loop - loop until final_string's length equals T.
260   // Which, T - the length of the kgram can get us there!
261   for (unsigned int a = 0; a < (T - (unsigned)_order); a++) {
262     // Call randk on the substring we're looking at.
263     // Note we want just _order long kgram to compare against.
264     return_char = randk(final_string.substr(a, _order));
265
266     // Add the return_char to final_string
267     final_string.push_back(return_char);
268
269     // Keep looping til it stops.
270   }
271
272   // When we get here, we're done. YAY.
273   return final_string;
274 }
275
276
277 /* Overload the stream insertion operator and display the internal
278  * state of the Markov Model. Print out the order, the alphabet,
279  * and the frequencies of the k-grams and k+1-grams.              */
280 std::ostream& operator<< (std::ostream &out, MarkovModel &mm) {
281   out << "\n_Order: " << mm._order << "\n";
282   out << "Alphabet: "<< mm._alphabet << "\n";
```

```
283
284   out << "Kgrams map: \n\n";
285
286   std::map<std::string, int>::iterator it;
287
288   for (it = mm._kgrams.begin(); it != mm._kgrams.end(); it++) {
289     out << it->first << "\t" << it->second << "\n";
290   }
291
292   return out;
293 }
```

PS7a: Kronos Time Clock Log Parsing Boot Parsing

The Assignment

This assignment asked us to use regular expressions to parse a log file from a Kronos InTouch time clock. To parse the file, we used Boost's regular expression library (regex), along with Boost's date/time libraries. This allowed us to create an output file that verifies whether a time clock successfully booted or failed to boot. I was able to accomplish this, and pass all of Bottlenose's tests successfully.

Key Concepts

The main idea for this assignment was regular expressions, and specifically how to use Boost's regex library to find matches in a string. A secondary idea was to get used to using date and time libraries to mess with dates and do some calculations on them. Boost's regex library was pretty easy to use – simply using boost::regex_match or boost::regex_search was enough to find a match against the regular expressions I created. The following expressions were what I used to match start boots, and successful completetions:

```
std::string start_string = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
start_string += "([0-9]{2}):([0-9]{2}):([0-9]{2}): \\(log.c.166\\) ";
start_string += "server started";
```

This is my regex to detect the boot statement. It finds a matching line with a time stamp, following by the words "(log.c.166) server started". Matching the regex against this whole statement made it easy to pull the date and time from the current line.

```
std::string end_string = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
end_string += "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}):INFO:oejs.";
end_string += "AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080";
```

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

What I Learned

I learned a ton about using regular expressions in this program – enough, that I feel like I could probably do some other parsing of files in the future if a job or future class requires it. I also can see a lot of practical uses for a program such as this – an intern (probably me in the future) could easily create parsing program to verify successful / failure code, or to double check that certain devices are working properly. It was also pretty handy to learn about date and time. Having used Boost's date and time libraries, I feel pretty good about using other libraries in the future that involve date and time. It has given me a solid introduction to working with dates – perhaps in the future I may need to do something that involves calculation dates and time.

Screenshots

NEED TO MAKE SCREENSHOT IN LINUX.

Source Code for PS7A

Makefile

```
1   # Makefile for PS7A
2   # Flags to save on typing all this out
3   CC = g++
4   CFLAGS = -g -Wall -Werror -std=c++0x -pedantic
5   SFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
6   Boost = -lboost_regex -lboost_date_time
7
8   # Make ps5a & a dummy tester
9   all:    ps7a
10
11  # PS7A executable
12  ps7a:   ps7a.o
13      $(CC) ps7a.o -o ps7a $(Boost)
14
15  # Object files
16  ps7a.o: ps7a.cpp
17      $(CC) -c ps7a.cpp $(CFLAGS)
18
19  # Cleanup object files
20  clean:
21      rm *.o
22      rm ps7a
23
```

main.cpp

```
1   /*
2    * Copyright 2015 Jason Downing
3    * All rights reserved.
4    * MIT Licensed - see http://opensource.org/licenses/MIT for details.
5    *
6    */
7   #include <boost/regex.hpp>
8   #include <iostream>
9   #include <fstream>
10  #include <string>
11  #include "boost/date_time/gregorian/gregorian.hpp"
12  #include "boost/date_time/posix_time/posix_time.hpp"
13
14  // Annoying to type these out.
15  using boost::gregorian::date;
16  using boost::gregorian::from_simple_string;
17  using boost::gregorian::date_period;
18  using boost::gregorian::date_duration;
19
20  using boost::posix_time::ptime;
21  using boost::posix_time::time_duration;
22
23  int main(int argc, const char* argv[]) {
24    // Make sure the user knows how to use our amazing log parser.
25    if (argc != 2) {
26      std::cout << "./ps7a device1_intouch.log\n";
27      return 0;
28    }
29
30    // Counters
31    int lines_scan = 1;     // START AT LINE 1 - not 0. opps.
32    int boot_success = 0;
33    int boot_total = 0;
34
35    // Some strings we need. File names, other stuff.
36    std::string file_name(argv[1]);
37    std::string output_name = file_name + ".rpt";
38    std::string report = "";
39    std::string boots = "";
40
41    // Begin time / End time strings.
42    std::string begin_date = "";
43    std::string end_date = "";
44
45    // A BUNCH OF STUFF I DON'T CARE.
46    std::string full_date;
47    int hours = 0;
48    int minutes = 0;
49    int seconds = 0;
50
51    ptime begin;
52    ptime end;
53
54    date date1;
```

```
55    date date2;
56
57    time_duration time_diff;
58
59    // Need to match against something like this:
60    // Start of boot: 2014-02-01 14:02:32: (log.c.166) server started
61    // Success if we find:
62    // "2014-01-26 09:58:04.362:INFO:oejs.AbstractConnector:Started
63    // SelectChannelConnector@0.0.0.0:9080"
64    std::string start_string = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
65    start_string += "([0-9]{2}):([0-9]{2}):([0-9]{2}): \\(log.c.166\\) ";
66    start_string += "server started";
67    std::string end_string = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
68    end_string += "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}):INFO:oejs.";
69    end_string += "AbstractConnector:Started
SelectChannelConnector@0.0.0.0:9080";
70
71    // Make two regexes
72    boost::regex start_regex(start_string, boost::regex::perl);
73    boost::regex end_regex(end_string);
74
75    // Use this for getting parts of the matched string.
76    boost::smatch sm;
77
78    // report += "Device boot report.\n\n";
79    // report += "InTouch log file: " + file_name + "\n";
80
81    // Read the file here and do stuff.
82    // Save stuff to a string with formatting to output later on.
83    std::string line;
84    std::ifstream file(file_name.c_str());
85
86    // Need to keep track of when we've found a start.
87    bool found_start = false;
88
89    if (file.is_open()) {
90      while (getline(file, line)) {
91        // We've got the current string here and can do stuff with it.
92
93        // Wipe the begin_time / end_time strings
94        begin_date.clear();
95        end_date.clear();
96
97        // Let's try and see if we found a start boot.
98        if (boost::regex_search(line, sm, start_regex)) {
99          // Get the start time, save it for later.
100         // Note: sm[0] is the ENTIRE match. We just want the date.
101         begin_date = sm[1] + "-" + sm[2] + "-" + sm[3];
102         begin_date += " " + sm[4] + ":" + sm[5] + ":" + sm[6];
103
104         full_date = sm[1] + "-" + sm[2] + "-" + sm[3];
105         date1 = date(from_simple_string(full_date));
106
107         hours = std::stoi(sm[4]);
108         minutes = std::stoi(sm[5]);
109         seconds = std::stoi(sm[6]);
110
```

```
111         begin = ptime(date1, time_duration(hours, minutes, seconds));
112
113         // We can use this begin_time for calculations later on.
114
115         // If we already found a start, then this is an incomplete boot.
116         if (found_start == true) {
117           boots += "**** Incomplete boot **** \n\n";
118         }
119
120         // Now we want to add this to the output string as boot start.
121         boots += "=== Device boot ===\n";
122         boots += std::to_string(lines_scan) + "(" + file_name + "): ";
123         boots += begin_date + " Boot Start\n";
124
125         boot_total++;
126
127         // Match this as a "found_start"
128         found_start = true;
129       }
130
131       // Or did we find a successful boot?
132        if (boost::regex_match(line, sm, end_regex)) {
133         // Get the end time, save it for later.
134         end_date = sm[1] + "-" + sm[2] + "-" + sm[3];
135         end_date += " " + sm[4] + ":" + sm[5] + ":" + sm[6];
136
137         full_date = sm[1] + "-" + sm[2] + "-" + sm[3];
138         date2 = date(from_simple_string(full_date));
139
140         hours = std::stoi(sm[4]);
141         minutes = std::stoi(sm[5]);
142         seconds = std::stoi(sm[6]);
143
144         end = ptime(date2, time_duration(hours, minutes, seconds));
145
146         // Add the end boot line and total time it took to get here.
147         boots += std::to_string(lines_scan) + "(" + file_name + "): ";
148         boots += end_date + " Boot Completed\n";
149
150         // Do some magic here and calculate the time it took to boot in ms
151         // Use the begin_time and the end_time variables.
152
153         // Time calculation stuff.
154         time_diff = end - begin;
155
156         // Now add the time difference.
157         boots += "\tBoot Time: ";
158         boots += std::to_string(time_diff.total_milliseconds()) + "ms \n\n";
159
160         boot_success++;
161
162         // Match found_start as false for the next loop.
163         found_start = false;
164        }
165
166       lines_scan++;
167     }
```

```
168       file.close();
169     }
170
171     // Add lines scanned to the report.
172     // report += "Lines scanned: " + std::to_string(lines_scan) + "\n\n";
173
174     // We can figure out success boots and fail boots now.
175     // report += "Device boot count: initiated = " + std::to_string(boot_total);
176     // report += ", completed: " + std::to_string(boot_success) + "\n\n\n";
177
178     // Need to remove an extra newline from the end of the boot report.
179     // This way the file will completely match the test cases on Bottlenose.
180     boots.erase(boots.end()-1);
181
182     // We should now add the boot reports to the end of this report.
183     report += boots;
184
185     // Now let's print out our report string. It has already been formatted
186     // correctly so it should be fairly simple to just dump it to stdio.
187     // std::cout << report;
188
189     // And we can even save this to a file with the extension ".rpt"
190     // which would be something like "device5_intouch.log.rpt"
191     // or filename + ".rpt"
192     std::ofstream out(output_name.c_str());
193     out << report;
194     out.close();
195
196     return 0;
197 }
```