University of Massachusetts Lowell

Computing IV Programming Portfolio

Spring 2017

Sam Pickell

COMP 2040

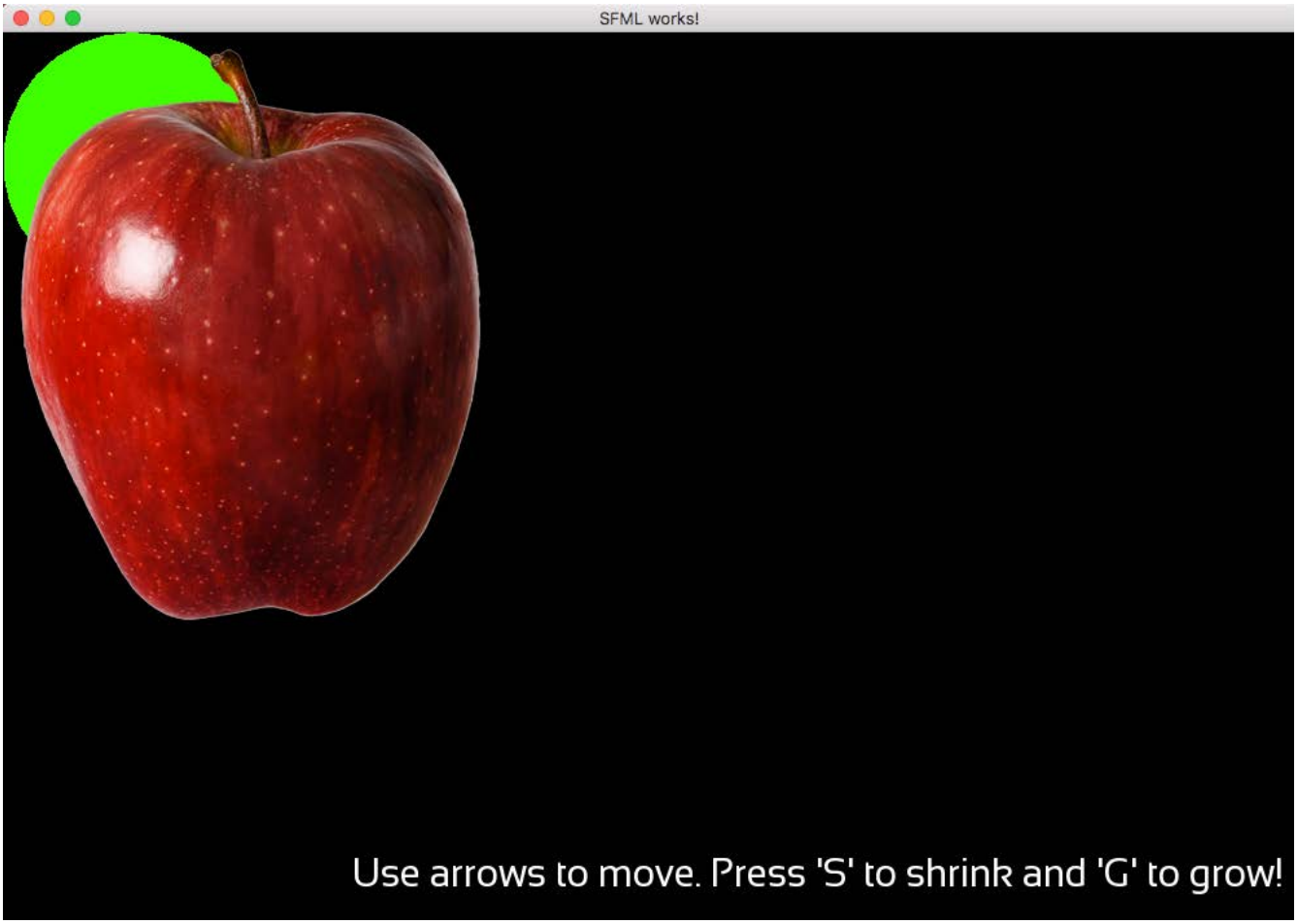
Dr. Rykalova

April 25, 2017

Contents:

PS0: Hello World with SFML

This assignment introduced me to the basics of SFML: outputting graphics to the screen, moving the graphics around, layering them, etc. It laid the groundwork for what future graphical SFML projects would entail and got me familiar with how to set up a window and how to draw to the screen. After reading much of the SFML documentation, I quickly learned that future programs were going to involve much more than the mathematical formulas I was used to coding; I was going to have to learn how to manipulate objects on the screen. This is the only program on this list that does not include a makefile as at the specifications did not require one.

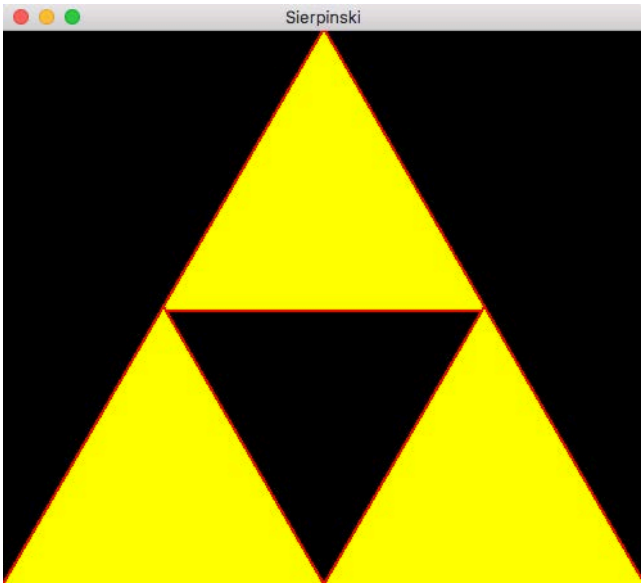


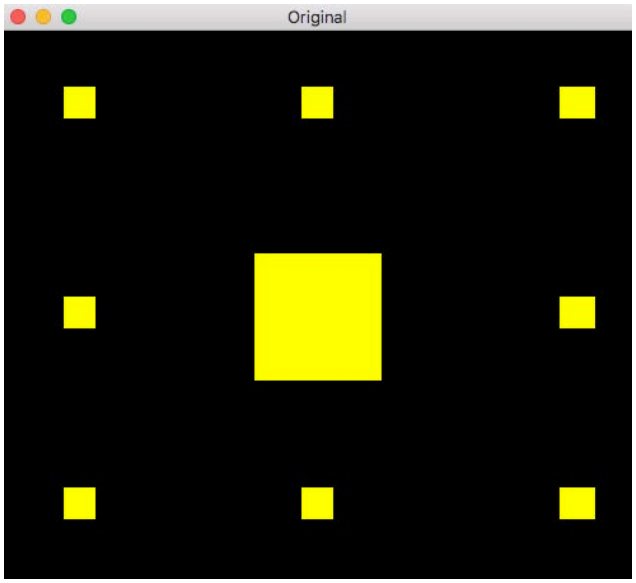
```
    1: #include <SFML/Graphics.hpp>
    2:
    3: using namespace sf;
    4: using namespace std;
    5:
    6: int main()
    7: {
    8:     int x = 0, y = 0;
    9:     float size_x = 1, size_y = 1, x_barrier = 500, y_barrier = 158;
   10:     Font font;
   11:     Text text("Use arrows to move. Press 'S' to shrink and 'G' to grow!", fo
nt, 30);
   12:     Texture texture;
   13:
   14:     font.loadFromFile("sansation.ttf");
   15:     text.setPosition(270, 630);
   16:     RenderWindow window(VideoMode(1000, 1000), "SFML works!");
   17:     window.setPosition(Vector2i(0,0));
   18:     CircleShape shape(100.f);
   19:     shape.setFillColor(Color::Green);
   20:
   21:     //Load a sprite
   22:     if (!texture.loadFromFile("sprite.png"))
   23:     {
   24:         return EXIT_FAILURE;
   25:     }
   26:     Sprite sprite(texture);
   27:
   28:     while (window.isOpen())
   29:     {
   30:         sprite.setPosition(x, y);
   31:         sprite.setScale(size_x, size_y);
   32:         Event event;
   33:         while (window.pollEvent(event))
   34:         {
   35:             if (event.type == Event::Closed)
   36:                 window.close();
   37:         }
   38:
   39:         if(Keyboard::isKeyPressed(Keyboard::Left) && x > 0)
   40:         {
   41:             x--;
   42:         }
   43:         if(Keyboard::isKeyPressed(Keyboard::Right) && x < x_barrier)
   44:         {
   45:             x++;
   46:         }
   47:         if(Keyboard::isKeyPressed(Keyboard::Up) && y > 0)
   48:         {
   49:             y--;
   50:         }
   51:         if(Keyboard::isKeyPressed(Keyboard::Down) && y < y_barrier)
   52:         {
   53:             y++;
   54:         }
   55:         if(Keyboard::isKeyPressed(Keyboard::S))
   56:         {
   57:             size_x -= .001;
   58:             size_y -= .001;
   59:             x_barrier += .5;
   60:             y_barrier += .5;
```

```
61:         }
62:         if(Keyboard::isKeyPressed(Keyboard::G))
63:         {
64:             size_x += .001;
65:             size_y += .001;
66:             x_barrier -= .5;
67:             y_barrier -= .5;
68:         }
69:
70:         window.clear();
71:         window.draw(shape);
72:         window.draw(sprite);
73:         window.draw(text);
74:         window.display();
75:     }
76:
77:     return 0;
78: }
```

PS1: Recursive Graphics

I was not successful in completing this project. This project was supposed to implement Sierpinski's Triangle with a depth and side given by the user. The main algorithm that was supposed to be used was recursion, where each set of three triangles was supposed to call itself while the depth was not met. I was also asked to create an original design, and I tried to implement Sierpinski's carpet which was like the triangle but was a rectangle instead. As mentioned earlier, I was unsuccessful in making the program work. I was able to make both Sierpinski and Original to work for N = 0 and N = 1, but after that, the images would get distorted. I did, however, learn a couple things because of this project. The first was how to make multiple executables with one makefile, something I had never done before. The second is that Xcode was not an environment that I could continue coding in, and that partitioning my hard drive and putting Ubuntu on it and coding in the command line was how I was going to continue coding in the class going forward. Both of those are very valuable skills that I am glad that I now have, and I do not think I would have learned had I completed this assignment successfully.





Makefile Thu Apr 20 17:40:49 2017 1

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=original.o maine.o
 4: S=sierpinski.o main.o
 5: P=sierpinski
 6: Q=original
 7: SFML= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 8: all: $(P) $(Q)
 9: $(P):$(S)
10:         $(C) -o $(P) $(S) $(SFML)
11:
12: $(Q):$(O)
13:         $(C) -o $(Q) $(O) $(SFML)
14:
15: $(E).o:
16:         $(C) -c $<  -o $@
17:
18: clean:
19:         rm $(O) $(P) $(Q) $(S)
20:
21: .PHONY: clean
```

```
    1: #include "sierpinski.hpp"
    2: #include <SFML/Window.hpp>
    3: #include <SFML/System.hpp>
    4: #include <iostream>
    5: #include <cmath>
    6:
    7: int main(int argc, char* argv[])
    8: {
    9:     /*
   10:     if (argc < 3 || argc > 4)
   11:     {
   12:         std::cout << "Sierpinski [recursion-depth] [side-length]" << std::en
dl;
   13:         return -1;
   14:     }
   15:
   16:     int depth = atoi(argv[1]);
   17:     int side = atoi(argv[2]);
   18:      */
   19:
   20:     int depth;
   21:     int side;
   22:
   23:     std::cout << "Enter depth: ";
   24:     std::cin >> depth;
   25:     std::cout << "Enter side: ";
   26:     std::cin >> side;
   27:
   28:     if (argc < 3 || argc > 4)
   29:     {
   30:         std::cout << "Sierpinski [side-length] [recursion-depth]" << std::en
dl;
   31:         return -1;
   32:     }
   33:
   34:     std::cout << "Depth: " << depth << std::endl;
   35:     std::cout << "Side: " << side << std::endl;
   36:
   37:     if (depth < 0)
   38:     {
   39:         std::cout << "Depth should be greater than 0" << std::endl;
   40:     }
   41:
   42:     Sierpinski obj(side, depth);
   43:
   44:
   45:     int window_height = (int)(.5*sqrt(3.)*(float)side);
   46:
   47:     sf::RenderWindow window(sf::VideoMode(side, window_height), "Sierpinski"
);
   48:
   49:     window.setFramerateLimit(1);
   50:
   51:     while(window.isOpen())
   52:     {
   53:         sf::Event event;
   54:         while(window.pollEvent(event))
   55:         {
   56:             if(event.type == sf::Event::Closed)
   57:             {
   58:                 window.close();
```

```
59:             }
60:         }
61:         window.clear();
62:         window.draw(obj);
63:         window.display();
64:     }
65:
66:     return 0;
67: }
```

sierpinski.hpp Mon Jan 30 17:26:16 2017 1

```
    1: #ifndef SIERPINSKI_H
    2: #define SIERPINSKI_H
    3: #include <SFML/Graphics.hpp>
    4: #include <vector>
    5:
    6:
    7: class Sierpinski : public sf::Drawable
    8: {
    9: public:
   10:     //Constructors
   11:
   12:     //define with side length and depth
   13:     Sierpinski(int size, int depth);
   14:
   15:     //set up a triangle
   16:     Sierpinski(float x1, float y1, float x2, float y2, float x3, float y3, i
nt depth, float size, float height);
   17:
   18:     //Destructor
   19:     ~Sierpinski();
   20:
   21:     //Functions
   22:
   23:
   24:
   25: private:
   26:
   27:     void virtual draw(sf::RenderTarget &target, sf::RenderStates states) con
st;
   28: //    float sierpinski_depth;
   29: //    float sierpinski_size;
   30: //    float sierpinski_height;
   31:
   32: };
   33: #endif
```

```
    1: #include "sierpinski.hpp"
    2: #include <cmath>
    3: #include <iostream>
    4:
    5: std::vector <sf::ConvexShape> triangle_vector;
    6: int count = 0;
    7:
    8: Sierpinski::Sierpinski(int size, int depth)
    9: {
   10:     float sierpinski_height = size * sqrt(3)/2;
   11: //    sierpinski_depth = depth;
   12:  //   sierpinski_size = size;
   13:
   14:
   15:     sf::Vector2f p1, p2, p3;
   16:     p1.x = size/2;
   17:     p1.y = 0;
   18:     p2.x = 0;
   19:     p2.y = sierpinski_height;
   20:     p3.x = size;
   21:     p3.y = sierpinski_height;
   22:
   23:     //Set Initial Triangle
   24:     sf::ConvexShape initial_triangle;
   25:     initial_triangle.setPointCount(3);
   26:     initial_triangle.setPoint(0, p1);
   27:     initial_triangle.setPoint(1, p2);
   28:     initial_triangle.setPoint(2, p3);
   29:     initial_triangle.setFillColor(sf::Color::Yellow);
   30:
   31:     triangle_vector.push_back(initial_triangle);
   32:
   33:     if (depth > 0)
   34:     {
   35:         triangle_vector.pop_back();
   36:         Sierpinski(p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, depth, size, sierpins
ki_height);
   37:     }
   38:
   39:
   40:
   41: }
   42:
   43:
   44: Sierpinski::Sierpinski(float x1, float y1, float x2, float y2, float x3, flo
at y3, int depth,
   45:                        float size, float height)
   46: {
   47:     /*if (depth not reached)
   48:     {
   49:         child (top, midleft (w/4) (h/2), midright (3w/4)(h/2))
   50:         child (midleft, left, middle (top w/2) (h))
   51:         child (midright, middle, right)
   52:     }
   53:      else
   54:      {
   55:         build triangle with current data
   56:      }
   57:      */
   58:
   59:
```

```
   60:
   61:      if (depth > 0)
   62:      {
   63:         depth--;
   64:
   65:         Sierpinski(x1, y1, size/4, height/2, ((3 * size)/4), height/2, depth
, size/4, height/2);
   66:         Sierpinski(size/4, height/2, x2, y2, x1, height, depth, size/4, heig
ht/2);
   67:         Sierpinski(((3 * size)/4), height/2, x1, height, x3, y3, depth, size
/4, height/2);
   68:
   69:      }
   70:      else
   71:      {
   72:          sf::Vector2f p1, p2, p3;
   73:          p1.x = x1;
   74:          p1.y = y1;
   75:          p2.x = x2;
   76:          p2.y = y2;
   77:          p3.x = x3;
   78:          p3.y = y3;
   79:
   80:         sf::ConvexShape triangle;
   81:         triangle.setPointCount(3);
   82:         triangle.setPoint(0, p1);
   83:         triangle.setPoint(1, p2);
   84:         triangle.setPoint(2, p3);
   85:         triangle.setFillColor(sf::Color::Yellow);
   86:          triangle.setOutlineColor(sf::Color::Red);
   87:          triangle.setOutlineThickness(2);
   88:
   89:         triangle_vector.push_back(triangle);
   90:          count++;
   91:      }
   92: }
   93:
   94:
   95: Sierpinski::~Sierpinski()
   96: {
   97:
   98: }
   99:
  100:
  101: void Sierpinski::draw(sf::RenderTarget &target, sf::RenderStates states) con
st
  102: {
  103:     for(unsigned int i = 0; i < triangle_vector.size(); i++)
  104:     {
  105:         target.draw(triangle_vector.at(i), states);
  106:     }
  107: }
  108:
  109:
  110:
  111:
```

```
    1: #ifndef ORIGINAL_H
    2: #define ORIGINAL_H
    3: #include <SFML/Graphics.hpp>
    4: #include <vector>
    5:
    6:
    7: class original : public sf::Drawable
    8: {
    9: public:
   10:     //Constructors
   11:
   12:     //define with side length and depth
   13:     original(int size, int depth);
   14:
   15:     //set up a shape
   16:     original(float x1, float y1, float x2, float y2, float x3, float y3, flo
at x4, float y4,
   17:              int depth, float size, float height);
   18:
   19:     //Destructor
   20:     ~original();
   21:
   22:     //Functions
   23:
   24:
   25:
   26: private:
   27:
   28:     void virtual draw(sf::RenderTarget &target, sf::RenderStates states) con
st;
   29:     //    float original_depth;
   30:     //    float original_size;
   31:     //    float original_height;
   32:
   33: };
   34: #endif
```

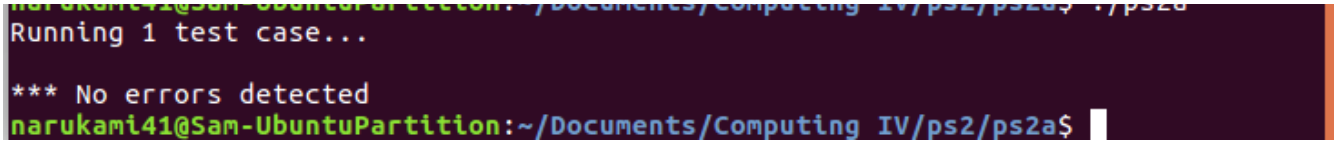
```
    1: #include "original.hpp"
    2: #include <SFML/Window.hpp>
    3: #include <cmath>
    4: #include <iostream>
    5:
    6: std::vector <sf::ConvexShape> square_vector;
    7: int count = 0;
    8:
    9: original::original(int size, int depth)
   10: {
   11:     float original_height = size/2;
   12:     //    original_depth = depth;
   13:     //   original_size = size;
   14:
   15:
   16:     sf::Vector2f p1, p2, p3, p4;
   17:     p1.x = size *.4;
   18:     p1.y = size *.35;
   19:     p2.x = size*.4;
   20:     p2.y = size*.55;
   21:     p3.x = size*.6;
   22:     p3.y = size*.55;
   23:     p4.x = size*.6;
   24:     p4.y = size*.35;
   25:
   26:     //Set Initial Triangle
   27:     sf::ConvexShape initial_square;
   28:     initial_square.setPointCount(4);
   29:     initial_square.setPoint(0, p1);
   30:     initial_square.setPoint(1, p2);
   31:     initial_square.setPoint(2, p3);
   32:     initial_square.setPoint(3, p4);
   33:     initial_square.setFillColor(sf::Color::Yellow);
   34:
   35:     square_vector.push_back(initial_square);
   36:
   37:     original(p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, depth, size, or
iginal_height);
   38:
   39: }
   40:
   41:
   42: original::original(float x1, float y1, float x2, float y2, float x3, float y
3, float x4, float y4,
   43:                     int depth, float size, float height)
   44: {
   45:         /*if (depth not reached)
   46:          {
   47:          child (top, midleft (w/4) (h/2), midright (3w/4)(h/2))
   48:          child (midleft, left, middle (top w/2) (h))
   49:          child (midright, middle, right)
   50:          }
   51:          else
   52:          {
   53:          build triangle with current data
   54:          }
   55:          */
   56:
   57:
   58:
   59:         if (depth > 0)
```

```
   60:         {
   61:             depth--;
   62:
   63:             original(x1/4, y1/4, x2/4, y2/4, x3/4, y3/4, x4/4, y4/4, depth,
size, height);
   64:
   65:             original((x1+size*.5)/1.9, y1/4, (x2+size*.5)/1.9, y2/4, (x3+siz
e*.5)/2.1, y3/4, (x4+size*.5)/2.1, y4/4, depth, size, height);
   66:
   67:             original((x1 * 2.2), y1/4, (x2 * 2.2), y2/4, (x3 * 1.56), y3/4,
(x4 * 1.56), y4/4,
   68:                      depth, size, height);
   69:
   70:             original(x1/4, y1/4 + size*.63, x2/4, y2/4 + size*.63, x3/4, y3/
4 + size*.63, x4/4,
   71:                      y4/4 + size*.63, depth, size, height);
   72:
   73:             original((x1+size*.5)/1.9, y1/4 + size*.63, (x2+size*.5)/1.9, y2
/4 + size*.63, (x3+size*.5)/2.1, y3/4 + size*.63, (x4+size*.5)/2.1, y4/4 + size*.63
, depth, size, height);
   74:
   75:             original((x1 * 2.2), y1/4 + size*.63, (x2 * 2.2), y2/4 + size*.6
3, (x3 * 1.56), y3/4 + size*.63, (x4 * 1.56), y4/4 + size*.63, depth, size, height)
;
   76:
   77:             original(x1/4, y1/4 + size*.33, x2/4, y2/4 + size*.33, x3/4, y3/
4 + size*.33, x4/4,
   78:                      y4/4 + size*.33, depth, size, height);
   79:
   80:             original((x1 * 2.2), y1/4 + size*.33, (x2 * 2.2), y2/4 + size*.3
3, (x3 * 1.56), y3/4 + size*.33, (x4 * 1.56), y4/4 + size*.33, depth, size, height)
;
   81:
   82:
   83:         }
   84:         else
   85:         {
   86:             sf::Vector2f p1, p2, p3, p4;
   87:             p1.x = x1;
   88:             p1.y = y1;
   89:             p2.x = x2;
   90:             p2.y = y2;
   91:             p3.x = x3;
   92:             p3.y = y3;
   93:             p4.x = x4;
   94:             p4.y = y4;
   95:
   96:             //Set Initial Triangle
   97:             sf::ConvexShape initial_square;
   98:             initial_square.setPointCount(4);
   99:             initial_square.setPoint(0, p1);
  100:             initial_square.setPoint(1, p2);
  101:             initial_square.setPoint(2, p3);
  102:             initial_square.setPoint(3, p4);
  103:             initial_square.setFillColor(sf::Color::Yellow);
  104:
  105:             square_vector.push_back(initial_square);
  106:         }
  107: }
  108:
  109:
```

```
110: original::~original()
111: {
112:
113: }
114:
115:
116: void original::draw(sf::RenderTarget &target, sf::RenderStates states) const
117: {
118:     for(int i = 0; i < square_vector.size(); i++)
119:     {
120:         target.draw(square_vector.at(i), states);
121:     }
122: }
123:
124:
125:
126:
```

PS2a: Linear Feedback Register Part A

The first of two parts for program two. This part of the program introduced me to the C++ Boost library, specifically the test portion of it. Boost testing allows the programmer to set up a serious of test cases to check and see if certain aspects of the program are working properly. I also set up a Linear Feedback Shift Register (LFSR for short) that would be implemented in the second part of this assignment, and tested in this part.



Makefile Mon Feb 06 10:36:27 2017 1

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=test.o LFSR.o
 4: P=ps2a
 5: BOOST= -lboost_unit_test_framework
 6: #SFML= -lsfml-graphics -lsfml-window -lsfml-system
 7: all: $(P)
 8: $(P):$(O)
 9:         $(C) -o $(P) $(O) $(BOOST)
10:
11: $(E).o:
12:         $(C) -c $<  -o $@
13:
14: clean:
15:         rm $(O) $(P)
```

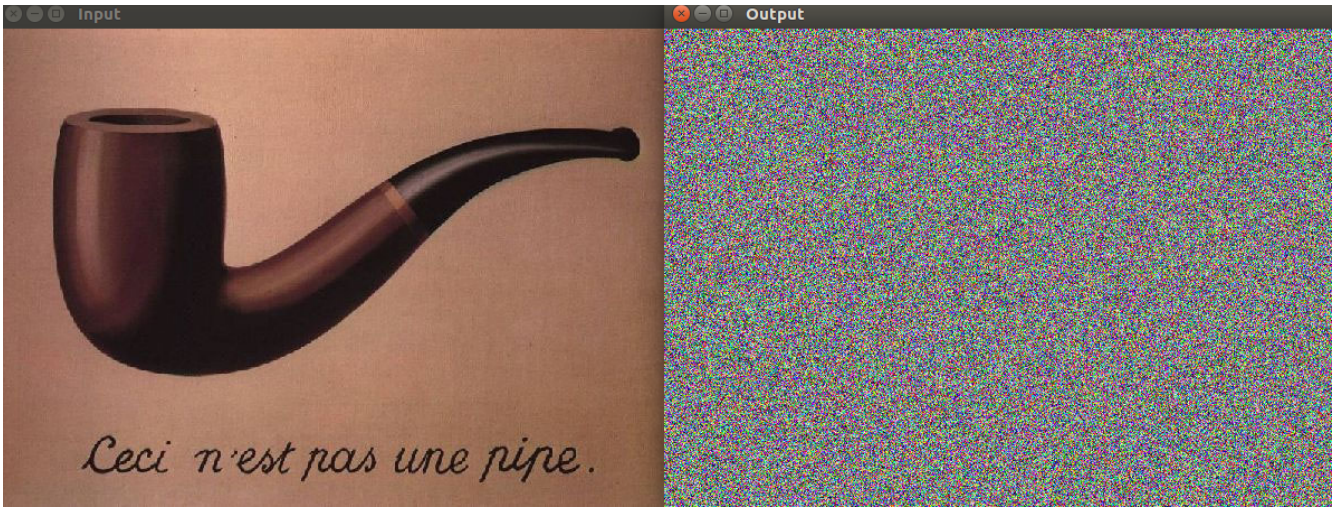
```
 1: #include "LFSR.hpp"
 2:
 3: #define BOOST_TEST_DYN_LINK
 4: #define BOOST_TEST_MODULE Main
 5: #include <boost/test/unit_test.hpp>
 6:
 7: BOOST_AUTO_TEST_CASE(fiveBitsTapAtTwo)
 8: {
 9:
10:   LFSR l("00111", 2);
11:   BOOST_REQUIRE(l.step() == 1);
12:   BOOST_REQUIRE(l.step() == 1);
13:   BOOST_REQUIRE(l.step() == 0);
14:   BOOST_REQUIRE(l.step() == 0);
15:   BOOST_REQUIRE(l.step() == 0);
16:   BOOST_REQUIRE(l.step() == 1);
17:   BOOST_REQUIRE(l.step() == 1);
18:   BOOST_REQUIRE(l.step() == 0);
19:
20:   LFSR l2("00111", 2);
21:   BOOST_REQUIRE(l2.generate(8) == 198);
22: }
```

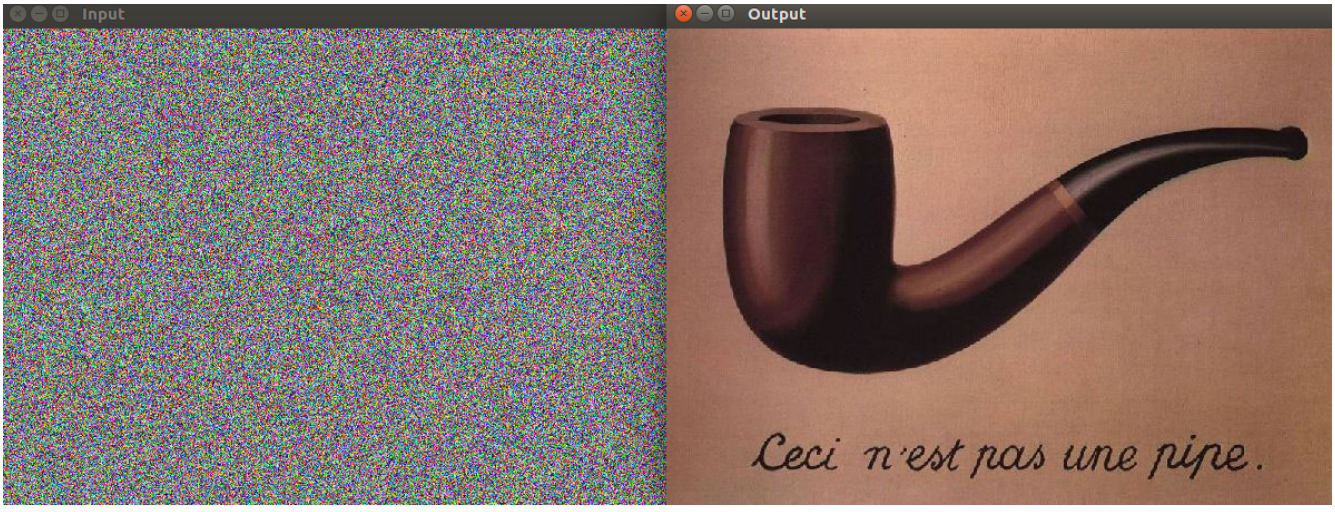
```
 1: #ifndef LFSR_HPP
 2: #define LFSR_HPP
 3:
 4: #include <iostream>
 5: #include <string>
 6:
 7: class LFSR
 8: {
 9:
10: public:
11:   LFSR(std::string user_seed, int user_tap);
12:   ~LFSR();
13:   int step();
14:   int generate(int k);
15:
16:   friend std::ostream& operator<< (std::ostream &out, LFSR &lfsr);
17:
18: private:
19:   std::string seed;
20:   int tap;
21:
22: };
23:
24:
25: #endif
```

```
 1: #include "LFSR.hpp"
 2:
 3: LFSR::LFSR(std::string user_seed, int user_tap)
 4: {
 5:   tap = user_tap;
 6:   seed = user_seed;
 7: }
 8:
 9: LFSR::~LFSR()
10: {
11:
12: }
13:
14: int LFSR::step()
15: {
16:
17:   int bit;
18:   char c_bit;
19:
20:   if (seed.at(0) == seed.at((seed.size()-1) - tap))
21:     {
22:       bit = 0;
23:       c_bit = '0';
24:     }
25:   else
26:     {
27:       bit = 1;
28:       c_bit = '1';
29:     }
30:
31:   for(unsigned int i = 0; i < (seed.size()-1); i++)
32:     {
33:       seed.at(i) = seed.at(i+1);
34:     }
35:   seed.at(seed.size()-1) = c_bit;
36:
37:   return bit;
38: }
39:
40: int LFSR::generate(int k)
41: {
42:   int gen = 0;
43:
44:   for (int i =0; i < k; i++)
45:     {
46:       gen = gen*2 + LFSR::step();
47:     }
48:
49:   return gen;
50: }
51:
52: std::ostream& operator<<(std::ostream &out,LFSR &lfsr)
53: {
54:   out << lfsr.seed;
55:
56:   return out;
57: }
```

PS2b: Linear Feedback Register Part B

The second part of program two. After setting up the LSFR object in the first part of this program, I was able to scramble and unscramble an image. The LSFR takes a seed and a key, and will generate different bit strings that are used to generate a random RBG color and change each pixel in the image to that color. After every pixel was scrambled, the picture becomes unrecognizable. However, if the same key is used while attempting to decode the image, each pixel will be reverted to what it was before the scrambling. This process is used in real life when encrypting and decrypting images and files, as the only way a user can generate the correct version of the file is if they have the proper key.





Makefile Sun Feb 12 11:48:50 2017 1

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=PhotoMagic.o LFSR.o
 4: P=PhotoMagic
 5: BOOST= -lboost_unit_test_framework
 6: SFML= -lsfml-graphics -lsfml-window -lsfml-system
 7: all: $(P)
 8: $(P):$(O)
 9:         $(C) -o $(P) $(O) $(BOOST) $(SFML)
10:
11: $(E).o:
12:         $(C) -c $<  -o $@
13:
14: clean:
15:         rm $(O) $(P)
```

```
    1: #include <SFML/System.hpp>
    2: #include <SFML/Window.hpp>
    3: #include <SFML/Graphics.hpp>
    4: #include <string>
    5: #include <cstdlib>
    6: #include "LFSR.hpp"
    7:
    8: void transform_method(sf::Image& image, sf::Color p, sf::Vector2u size, LFSR
 l);
    9:
   10: int main(int argc, char* argv[])
   11: {
   12:         //Read in Command Line Arguments
   13:         std::string user_image1 = argv[1];
   14:         std::string user_image2 = argv[2];
   15:         int tap = atoi(argv[4]);
   16:         LFSR l(argv[3], tap);
   17:
   18:         //Load Images
   19:         sf::Image image;
   20:         sf::Image image1;
   21:         if (!image.loadFromFile(user_image1))
   22:           {
   23:             return -1;
   24:           }
   25:         if (!image1.loadFromFile(user_image1))
   26:           {
   27:             return -1;
   28:           }
   29:
   30:         //Create pixel variable
   31:         sf::Color p;
   32:
   33:         //Set Size
   34:         sf::Vector2u size = image1.getSize();
   35:
   36:         //Transform the Images
   37:         transform_method(image1, p, size, l);
   38:
   39:         //Save Image
   40:         if (!image1.saveToFile(user_image2))
   41:           {
   42:             return -1;
   43:           }
   44:
   45:         //After transform
   46:         sf::Image image2;
   47:         if (!image2.loadFromFile(user_image2))
   48:           {
   49:             return -1;
   50:           }
   51:
   52:         //Generate Windows
   53:         sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Input");
   54:         sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "Output");
   55:
   56:         //Load Textures
   57:         sf::Texture texture1;
   58:         texture1.loadFromImage(image);
   59:         sf::Texture texture2;
   60:         texture2.loadFromImage(image2);
```

```
   61:
   62:         //Set Sprites
   63:         sf::Sprite sprite1;
   64:         sprite1.setTexture(texture1);
   65:
   66:         sf::Sprite sprite2;
   67:         sprite2.setTexture(texture2);
   68:
   69:
   70:
   71:         //Generate Windows
   72:         while (window1.isOpen() && window2.isOpen())
   73:         {
   74:                 sf::Event event;
   75:                 while (window1.pollEvent(event))
   76:                 {
   77:                     if (event.type == sf::Event::Closed)
   78:                       {
   79:                         window1.close();
   80:                       }
   81:                 }
   82:
   83:                 while (window2.pollEvent(event))
   84:                 {
   85:                     if (event.type == sf::Event::Closed)
   86:                       {
   87:                         window2.close();
   88:                       }
   89:                 }
   90:
   91:                 window1.clear(sf::Color::White);
   92:                 window1.draw(sprite1);
   93:                 window1.display();
   94:
   95:                 window2.clear(sf::Color::White);
   96:                 window2.draw(sprite2);
   97:                 window2.display();
   98:         }
   99:
  100:         return 0;
  101: }
  102:
  103: void transform_method(sf::Image& image, sf::Color p, sf::Vector2u size, LFSR
 1)
  104: {
  105:   for (unsigned int x = 0; x<size.x; x++)
  106:   {
  107:     for (unsigned int y = 0; y< size.y; y++)
  108:       {
  109:         int new_bit1, new_bit2, new_bit3;
  110:
  111:         new_bit1 = l.generate(8);
  112:         new_bit2 = l.generate(8);
  113:         new_bit3 = l.generate(8);
  114:
  115:         p = image.getPixel(x, y);
  116:         p.r = p.r ^ new_bit1;
  117:         p.g = p.g ^ new_bit2;
  118:         p.b = p.b ^ new_bit3;
  119:         image.setPixel(x, y, p);
  120:       }
```

```
121:    }
122: }
```

```
 1: #ifndef LFSR_HPP
 2: #define LFSR_HPP
 3:
 4: #include <iostream>
 5: #include <string>
 6:
 7: class LFSR
 8: {
 9:
10: public:
11:   LFSR(std::string user_seed, int user_tap);
12:   ~LFSR();
13:   int step();
14:   int generate(int k);
15:
16:   friend std::ostream& operator<< (std::ostream &out, LFSR &lfsr);
17:
18: private:
19:   std::string seed;
20:   int tap;
21:
22: };
23:
24:
25: #endif
```

```
 1: #include "LFSR.hpp"
 2:
 3: LFSR::LFSR(std::string user_seed, int user_tap)
 4: {
 5:   tap = user_tap;
 6:   seed = user_seed;
 7: }
 8:
 9: LFSR::~LFSR()
10: {
11:
12: }
13:
14: int LFSR::step()
15: {
16:
17:   int bit;
18:   char c_bit;
19:
20:   if (seed.at(0) == seed.at((seed.size()-1) - tap))
21:     {
22:       bit = 0;
23:       c_bit = '0';
24:     }
25:   else
26:     {
27:       bit = 1;
28:       c_bit = '1';
29:     }
30:
31:   for(unsigned int i = 0; i < (seed.size()-1); i++)
32:     {
33:       seed.at(i) = seed.at(i+1);
34:     }
35:   seed.at(seed.size()-1) = c_bit;
36:
37:   return bit;
38: }
39:
40: int LFSR::generate(int k)
41: {
42:   int gen = 0;
43:
44:   for (int i =0; i < k; i++)
45:     {
46:       gen = gen*2 + LFSR::step();
47:     }
48:
49:   return gen;
50: }
51:
52: std::ostream& operator<<(std::ostream &out,LFSR &lfsr)
53: {
54:   out << lfsr.seed;
55:
56:   return out;
57: }
```

PS3a: N-Body Simulation Part A

The first part of program three. This part was designed to set up the graphics for an N-Body simulation; the physics are added in the next part. I implemented a vector that contained all of the planets and the sun of the inner part of the solar system, and loaded a space background. Otherwise, the rest of this assignment was laying out the groundwork for the next part of the assignment where the actual simulation took place.

Makefile Sun Feb 19 15:29:10 2017 1

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=main.o Body.o
 4: P=NBody
 5: SFML= -lsfml-graphics -lsfml-window -lsfml-system
 6: all: $(P)
 7: $(P):$(O)
 8:         $(C) -o $(P) $(O) $(SFML)
 9:
10: $(E).o:
11:         $(C) -c $<  -o $@
12:
13: clean:
14:         rm $(O) $(P)
```

main.cpp Mon Feb 20 13:56:35 2017 1

```
 1: #include "Body.hpp"
 2: #include <vector>
 3:
 4: int main(int argc, char* argv[])
 5: {
 6:   sf::Image backdrop;
 7:
 8:   if(!backdrop.loadFromFile("starfield.jpg"))
 9:     {
10:       return -1;
11:     }
12:
13:   //Set up the universe
14:   std::vector<Body*> v_bodies;
15:   int number_of_bodies;
16:   double universe_size;
17:   sf::Vector2u size = backdrop.getSize();
18:
19:   std::cin >> number_of_bodies;
20:   std::cin >> universe_size;
21:
22:
23:   //Create the celestial bodies and put them into the vector
24:   for(int i = 0; i < number_of_bodies; i++)
25:     {
26:       Body* body = new Body();
27:       v_bodies.push_back(body);
28:     }
29:
30:   for(int i = 0; i < number_of_bodies; i++)
31:     {
32:       std::cin >> (*(v_bodies.at(i)));
33:       v_bodies.at(i)->set_radius(universe_size);
34:       v_bodies.at(i)->set_window(size.x);
35:       v_bodies.at(i)->update_pixel_pos();
36:     }
37:
38:   sf::Texture texture_drop;
39:   texture_drop.loadFromImage(backdrop);
40:
41:   sf::Sprite sprite_drop;
42:   sprite_drop.setTexture(texture_drop);
43:
44:
45:   sf::RenderWindow window(sf::VideoMode(size.x, size.y),
46:                           "NBody Program");
47:
48:   while(window.isOpen())
49:     {
50:       sf::Event event;
51:       while(window.pollEvent(event))
52:         {
53:           if(event.type == sf::Event::Closed)
54:             {
55:               window.close();
56:             }
57:         }
58:
59:       window.clear();
60:       window.draw(sprite_drop);
61:       for(int i = 0; i < number_of_bodies; i++)
```

```
62:          {
63:            window.draw(*(v_bodies.at(i)));
64:          }
65:        window.display();
66:      }
67:
68:    return 0;
69: }
```

Body.hpp Mon Feb 20 13:01:41 2017 1

```
 1: #ifndef BODY_HPP
 2: #define BODY_HPP
 3:
 4: #include <iostream>
 5: #include <SFML/Graphics.hpp>
 6: #include <SFML/Window.hpp>
 7: #include <SFML/System.hpp>
 8: #include <string>
 9:
10: class Body : public sf::Drawable
11: {
12: public:
13:
14:   Body();
15:   Body(double x, double y, double vel_x, double vel_y, double user_mass,
16:        std::string u_filename);
17:   ~Body();
18:
19:   //Accessors
20:   double get_xpos();
21:   double get_ypos();
22:
23:   //Mutators
24:   void set_radius(double rad);
25:   void set_window(int size);
26:
27:   void update_pixel_pos();
28:
29:   friend std::istream& operator >> (std::istream &input, Body &B);
30:
31: private:
32:
33:   virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
;
34:
35:   double xpos, ypos, velocity_x, velocity_y, mass, univ_rad;
36:   sf::Texture texture;
37:   sf::Sprite sprite;
38:   int window_size;
39:   std::string filename;
40:
41: };
42:
43:
44:
45: #endif
```

```
 1: #include "Body.hpp"
 2: #include <cstdlib>
 3:
 4: Body::Body()
 5: {
 6:
 7: }
 8:
 9: Body::Body(double x, double y, double vel_x, double vel_y,
10:            double user_mass, std::string u_filename)
11: {
12:   //initialize variables
13:   xpos = x;
14:   ypos = y;
15:   velocity_x = vel_x;
16:   velocity_y = vel_y;
17:   mass = user_mass;
18:   filename = u_filename;
19:
20:   //Load image
21:   sf::Image image;
22:   if(!image.loadFromFile(filename))
23:     {
24:       exit(1);
25:     }
26:
27:   texture.loadFromImage(image);
28:   sprite.setTexture(texture);
29: }
30:
31: Body::~Body()
32: {
33:
34:
35: }
36:
37: double Body::get_xpos()
38: {
39:   return xpos;
40: }
41:
42: double Body::get_ypos()
43: {
44:   return ypos;
45: }
46:
47: void Body::set_radius(double rad)
48: {
49:   univ_rad = rad;
50: }
51:
52: void Body::set_window(int size)
53: {
54:   window_size = size;
55: }
56:
57: void Body::update_pixel_pos()
58: {
59:   double percent_x, percent_y;
60:
61:   percent_x = (xpos + univ_rad) / (2 * univ_rad);
```

```
   62:   percent_y = (ypos + univ_rad) / (2 * univ_rad);
   63:
   64:   sprite.setPosition((window_size * percent_x),
   65:                       ((window_size * percent_y)));
   66: }
   67:
   68: std::istream& operator >> (std::istream &input, Body &B)
   69: {
   70:   input >> B.xpos >> B.ypos >> B.velocity_x >> B.velocity_y >>
   71:     B.mass >> B.filename;
   72:
   73:   //Load image
   74:   sf::Image image;
   75:   if(!image.loadFromFile(B.filename))
   76:     {
   77:       exit(1);
   78:     }
   79:
   80:   B.texture.loadFromImage(image);
   81:   B.sprite.setTexture(B.texture);
   82:
   83:
   84:
   85:   return input;
   86: }
   87:
   88: void Body::draw(sf::RenderTarget &target, sf::RenderStates states) const
   89: {
   90:   target.draw(sprite, states);
   91: }
```

PS3b: N-Body Simulation Part B

Part two of program three. This portion of the program implemented the physics involved with an N-Body diagram. I really enjoyed programming this part because this was the first time I felt that I was doing somewhat advanced level programming. I was used to programming functions and algorithms and never really saw my programs move before. I was able to calculate velocity and acceleration and the planets in the simulation began acting upon one another. Programming audio into this assignment was a fun touch, as it made all of the failures along the way entertaining and the final product all the more satisfying.

Makefile Thu Mar 02 22:42:03 2017 1

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=main.o Body.o
 4: P=NBody
 5: SFML= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 6: all: $(P)
 7:
 8: $(P):$(O)
 9:         $(C) -o $(P) $(O) $(SFML)
10:
11: $(E).o:
12:         $(C) -c $<  -o $@
13:
14: clean:
15:         rm $(O) $(P)
```

```
 1: #include "Body.hpp"
 2: #include <SFML/Audio.hpp>
 3:
 4: int main(int argc, char* argv[])
 5: {
 6:   sf::SoundBuffer buffer;
 7:   sf::Sound sound;
 8:   sf::Image backdrop;
 9:   double limit = atof(argv[1]);
10:   double interval = atof(argv[2]);
11:   double t = interval;
12:
13:   //Load backdrop
14:   if(!backdrop.loadFromFile("starfield.jpg"))
15:     {
16:       return -1;
17:     }
18:
19:   //Load Music
20:   if(!buffer.loadFromFile("2001.ogg"))
21:     {
22:       return -1;
23:     }
24:   sound.setBuffer(buffer);
25:   sound.play();
26:
27:
28:   //Set up the universe
29:   std::vector<Body*> v_bodies;
30:   int number_of_bodies;
31:   double universe_size;
32:   sf::Vector2u size = backdrop.getSize();
33:
34:   std::cin >> number_of_bodies;
35:   std::cin >> universe_size;
36:
37:
38:   //Create the celestial bodies and put them into the vector
39:   for(int i = 0; i < number_of_bodies; i++)
40:     {
41:       Body* body = new Body();
42:       v_bodies.push_back(body);
43:     }
44:
45:   for(int i = 0; i < number_of_bodies; i++)
46:     {
47:       std::cin >> (*(v_bodies.at(i)));
48:       v_bodies.at(i)->set_radius(universe_size);
49:       v_bodies.at(i)->set_window(size.x);
50:       v_bodies.at(i)->update_pixel_pos();
51:     }
52:
53:   sf::Texture texture_drop;
54:   texture_drop.loadFromImage(backdrop);
55:
56:   sf::Sprite sprite_drop;
57:   sprite_drop.setTexture(texture_drop);
58:
59:
60:   sf::RenderWindow window(sf::VideoMode(size.x, size.y),
61:                           "NBody Program");
```

```
   62:
   63:   while(window.isOpen() && t <= limit)
   64:   {
   65:       sf::Event event;
   66:       while(window.pollEvent(event))
   67:         {
   68:           if(event.type == sf::Event::Closed)
   69:             {
   70:               window.close();
   71:             }
   72:         }
   73:
   74:       window.clear();
   75:       window.draw(sprite_drop);
   76:       for(int i = 0; i < number_of_bodies; i++)
   77:         {
   78:               v_bodies.at(i)->step(interval, v_bodies);
   79:               v_bodies.at(i)->update_pixel_pos();
   80:               window.draw(*(v_bodies.at(i)));
   81:         }
   82:       window.display();
   83:       t += interval;
   84:     }
   85:
   86:
   87:   std::cout << number_of_bodies << std::endl;
   88:   std::cout << universe_size << std::endl;
   89:
   90:   for(unsigned int i = 0; i < v_bodies.size(); i++)
   91:     {
   92:       std::cout << v_bodies.at(i)->get_xpos() << " " <<  v_bodies.at(i)->get
_ypos() << " " <<
   93:         v_bodies.at(i)->get_velocity_x() << " " << v_bodies.at(i)->get_veloc
ity_y() << " " <<
   94:         v_bodies.at(i)->get_mass() << " " << v_bodies.at(i)->get_filename()
<< std::endl;
   95:     }
   96:
   97:   return 0;
   98: }
```

Body.hpp Thu Mar 02 21:30:43 2017 1

```
    1: #ifndef BODY_HPP
    2: #define BODY_HPP
    3:
    4: #include <iostream>
    5: #include <SFML/Graphics.hpp>
    6: #include <SFML/Window.hpp>
    7: #include <SFML/System.hpp>
    8: #include <vector>
    9: #include <string>
   10:
   11: class Body : public sf::Drawable
   12: {
   13: public:
   14:
   15:   Body();
   16:   Body(double x, double y, double vel_x, double vel_y, double user_mass,
   17:        std::string u_filename);
   18:   ~Body();
   19:
   20:   //Accessors
   21:   double get_xpos();
   22:   double get_ypos();
   23:   double get_velocity_x();
   24:   double get_velocity_y();
   25:   double get_mass();
   26:   std::string get_filename();
   27:
   28:   //Mutators
   29:   void add_to_xvel(double x);
   30:   void add_to_yvel(double y);
   31:   void step(double t, std::vector<Body*> v_bodies);
   32:   void set_radius(double rad);
   33:   void set_window(int size);
   34:
   35:   void update_pixel_pos();
   36:
   37:   friend std::istream& operator >> (std::istream &input, Body &B);
   38:
   39: private:
   40:
   41:   virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
;
   42:
   43:   double xpos, ypos, velocity_x, velocity_y, mass, univ_rad, F_x, F_y;
   44:   sf::Texture texture;
   45:   sf::Sprite sprite;
   46:   int window_size;
   47:   std::string filename;
   48:
   49: };
   50:
   51:
   52:
   53: #endif
```

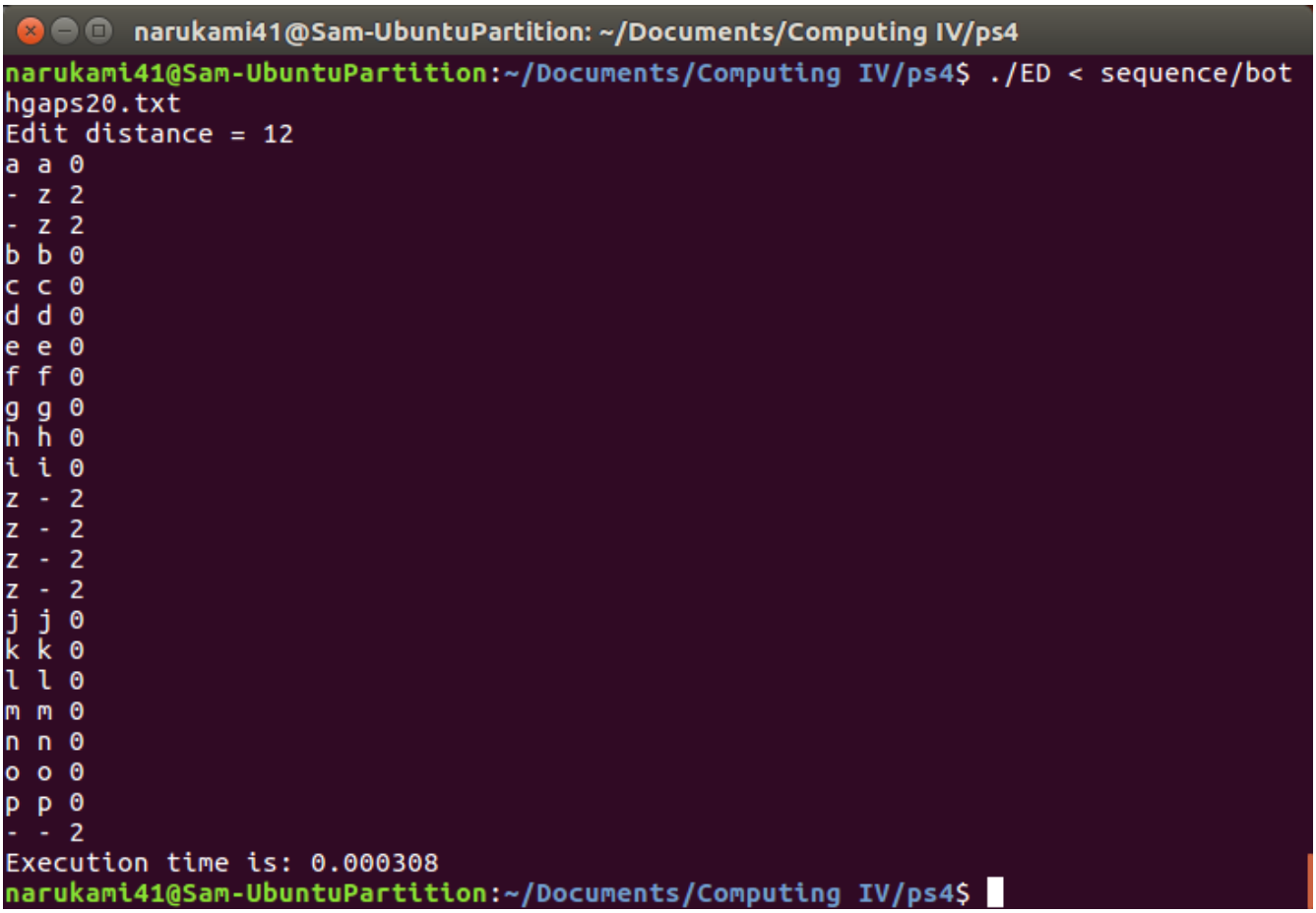
```
 1: #include "Body.hpp"
 2: #include <cstdlib>
 3: #include <cmath>
 4:
 5: Body::Body()
 6: {
 7:
 8: }
 9:
10: Body::Body(double x, double y, double vel_x, double vel_y,
11:            double user_mass, std::string u_filename)
12: {
13:   //initialize variables
14:   xpos = x;
15:   ypos = y;
16:   velocity_x = vel_x;
17:   velocity_y = vel_y;
18:   mass = user_mass;
19:   filename = u_filename;
20:
21:   //Load image
22:   sf::Image image;
23:   if(!image.loadFromFile(filename))
24:     {
25:       exit(1);
26:     }
27:
28:   texture.loadFromImage(image);
29:   sprite.setTexture(texture);
30: }
31:
32: Body::~Body()
33: {
34:
35:
36: }
37:
38: double Body::get_xpos()
39: {
40:   return xpos;
41: }
42:
43: double Body::get_ypos()
44: {
45:   return ypos;
46: }
47:
48: double Body::get_velocity_x()
49: {
50:   return velocity_x;
51: }
52:
53: double Body::get_velocity_y()
54: {
55:   return velocity_y;
56: }
57:
58: double Body::get_mass()
59: {
60:   return mass;
61: }
```

```
 62:
 63: std::string Body::get_filename()
 64: {
 65:   return filename;
 66: }
 67:
 68: void Body::add_to_xvel(double x)
 69: {
 70:   velocity_x += x;
 71: }
 72:
 73: void Body::add_to_yvel(double y)
 74: {
 75:   velocity_y += y;
 76: }
 77:
 78: void Body::step(double t, std::vector<Body*> v_bodies)
 79: {
 80:   double r, F, a_x, a_y, d_x, d_y;
 81:   const double G = 6.67e-11;
 82:
 83:   for(unsigned int i = 0; i < v_bodies.size(); i++)
 84:   {
 85:       if (v_bodies.at(i)->get_filename() != filename)
 86:         {
 87:           //Step 1
 88:           d_x = xpos - v_bodies.at(i)->get_xpos();
 89:           d_y = ypos - v_bodies.at(i)->get_ypos();
 90:
 91:           r = sqrt((pow(d_x, 2)) + (pow(d_y, 2)));
 92:           F = (G*mass*v_bodies.at(i)->get_mass()) / (pow(r, 2));
 93:           F_x = F*(d_x / r);
 94:           F_y = F*(d_y / r);
 95:           //End Step 1
 96:
 97:           //Step2
 98:           a_x = F_x / mass;
 99:           a_y = F_y / mass;
100:
101:           velocity_x += (t*a_x);
102:           velocity_y += (t*a_y);
103:
104:
105:           //End Step 2
106:         }
107:    }
108:
109:   xpos -= (t*velocity_x);
110:   ypos -= (t*velocity_y);
111: }
112:
113: void Body::set_radius(double rad)
114: {
115:   univ_rad = rad;
116: }
117:
118: void Body::set_window(int size)
119: {
120:   window_size = size;
121: }
122:
```

```
123: void Body::update_pixel_pos()
124: {
125:   double percent_x, percent_y;
126:
127:   percent_x = (xpos + univ_rad) / (2 * univ_rad);
128:   percent_y = (ypos + univ_rad) / (2 * univ_rad);
129:
130:   sprite.setPosition((window_size * percent_x),
131:                      ((window_size * percent_y)));
132: }
133:
134: std::istream& operator >> (std::istream &input, Body &B)
135: {
136:   input >> B.xpos >> B.ypos >> B.velocity_x >> B.velocity_y >>
137:     B.mass >> B.filename;
138:
139:   //Load image
140:   sf::Image image;
141:   if(!image.loadFromFile(B.filename))
142:     {
143:       exit(1);
144:     }
145:
146:   B.texture.loadFromImage(image);
147:   B.sprite.setTexture(B.texture);
148:
149:
150:
151:   return input;
152: }
153:
154: void Body::draw(sf::RenderTarget &target, sf::RenderStates states) const
155: {
156:   target.draw(sprite, states);
157: }
```

PS4: Edit Distance

Edit distance is based on DNA sequence alignment and finding the most efficient pairings between two strands of DNA (or in my case, two strings). When a character is matched with itself, the cost is zero, if the characters mismatch the cost is one, and if the character is matched with a space, the cost is two. This was probably the hardest assignment that I completed, as each function within the edit distance class took a very long time to program. I started by generating a matrix to see if the correct costs were being calculated, and then went on to program the matching portion. This program taught me that there are many more applications to programming than I had initially thought. Programming extends to the medical field in more ways than just hospital devices or computer programs, and how important programs like these are in scientific research.



Makefile Wed Mar 29 17:50:28 2017 1

```
    1: C=g++ -g -Wall --std=c++98 -Werror
    2: E=.cpp
    3: O=main.o Editdistance.o
    4: P=ED
    5: SFML= -lsfml-system
    6: all: $(P)
    7: $(P):$(O)
    8:          $(C) -o $(P) $(O) $(SFML)
    9:
   10: $(E).o:
   11:          $(C) -c $<  -o $@
   12:
   13: clean:
   14:          rm $(O) $(P)
```

main.cpp Wed Mar 29 22:30:19 2017 1

```
 1: #include "Editdistance.hpp"
 2: #include <SFML/System.hpp>
 3:
 4: int main(int argc, char* argv[])
 5: {
 6:   sf::Clock clock;
 7:   sf::Time t;
 8:   std::string test_1, test_2, align;
 9:
10:   std::cin >> test_1;
11:   std::cin >> test_2;
12:
13:   EditDistance ED(test_1, test_2);
14:
15:   int opt = ED.OptDistance();
16:
17:   std::cout << "Edit distance = " << opt << std::endl;
18:
19:   align = ED.Alignment();
20:
21:   std::cout << align;
22:
23:   t = clock.getElapsedTime();
24:
25:   std::cout << "Execution time is: " << t.asSeconds() << std::endl;
26:
27:   return 0;
28: }
```

```
 1: #ifndef EDITDISTANCE_HPP
 2: #define EDITDISTANCE_HPP
 3:
 4: #include <string>
 5: #include <vector>
 6: #include <iostream>
 7:
 8: class EditDistance
 9: {
10:  public:
11:   EditDistance(std::string string_1, std::string string_2);
12:   ~EditDistance();
13:   int penalty(char a, char b);
14:   int min(int a, int b, int c);
15:   int OptDistance();
16:   std::string Alignment();
17:
18:  private:
19:   std::vector< std::vector< int > > data;
20:   std::string x, y;
21: };
22: #endif
```

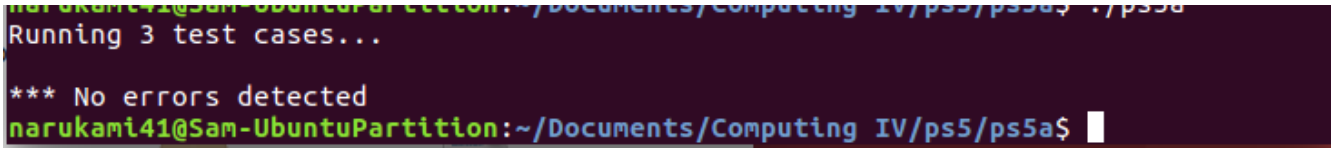
```
#include "Editdistance.hpp"

EditDistance::EditDistance(std::string string_1, std::string string_2)
{
  x = string_1;
  y = string_2;
  x.append("-");
  y.append("-");

  std::vector< std::vector< int > > my_data(string_2.size() + 1,
                  std::vector<int> (string_1.size() + 1, -1));
  data = my_data;
}

EditDistance::~EditDistance()
{

}

int EditDistance::penalty(char a, char b)
{
  if(a == b)
    {
      return 0;
    }
  else if ((a != b) && (a == '-' || b == '-'))
    {
      return 2;
    }
  else
    {
      return 1;
    }
}

int EditDistance::min(int a, int b, int c)
{
  if(a <= b)
    {
      if(a <= c)
        {
          return a;
        }
      else
        {
          return c;
        }
    }
  else
    {
      if(b <= c)
        {
          return b;
        }
      else
        {
          return c;
        }
    }
}

```

```
 62: int EditDistance::OptDistance()
 63: {
 64:   // This handles the outsides
 65:   int count = 0;
 66:   for(int i = x.size() - 1; i >= 0; i--)
 67:     {
 68:
 69:       data.at(y.size() - 1).at(i) = count;
 70:       count += 2;
 71:     }
 72:   count = 0;
 73:   for(int i = y.size() - 1; i >= 0; i--)
 74:     {
 75:       data.at(i).at(x.size() - 1) = count;
 76:       count += 2;
 77:     }
 78:
 79:   //this handles the inside
 80:   for(int i = x.size() - 2; i >= 0; i--)
 81:     {
 82:
 83:       for(int j = y.size() - 2; j >= 0; j--)
 84:         {
 85:           data.at(j).at(i) = min(
 86:               data.at(j+1).at(i+1) + penalty(x.at(i), y.at(j)),
 87:               data.at(j).at(i+1) + 2,
 88:               data.at(j+1).at(i) + 2);
 89:         }
 90:     }
 91:
 92:
 93:   return data.at(0).at(0);
 94: }
 95:
 96: std::string EditDistance::Alignment()
 97: {
 98:   std::string z;
 99:   unsigned int i = 0;
100:   unsigned int j = 0;
101:
102:   while(i < y.size() -1 && j < x.size() - 1)
103:   {
104:     if(data.at(i).at(j) == data.at(i+1).at(j+1) + penalty(x.at(j),y.at(i)))
105:     {
106:       z.push_back(x.at(j));
107:       z.push_back(' ');
108:       z.push_back(y.at(i));
109:       z.push_back(' ');
110:       z.push_back(penalty(x.at(j),y.at(i)) + '0');
111:       z.push_back('\n');
112:       i++;
113:       j++;
114:     }
115:     else if(data.at(i).at(j) ==  data.at(i).at(j+1) +2)
116:     {
117:       z.push_back(x.at(j));
118:       z.push_back(' ');
119:       z.push_back('-');
120:       z.push_back(' ');
121:       z.push_back('2');
122:       z.push_back('\n');
```

```
123:         j++;
124:       }
125:       else if(data.at(i).at(j) ==  data.at(i+1).at(j) +2)
126:       {
127:         z.push_back('-');
128:         z.push_back(' ');
129:         z.push_back(y.at(i));
130:         z.push_back(' ');
131:         z.push_back('2');
132:         z.push_back('\n');
133:         i++;
134:       }
135:    }
136:
137:    if(j <x.size())
138:    {
139:        z.push_back(x.at(j));
140:        z.push_back(' ');
141:        z.push_back('-');
142:        z.push_back(' ');
143:        z.push_back('2');
144:        z.push_back('\n');
145:        j++;
146:    }
147:    else if(i < y.size())
148:    {
149:        z.push_back('-');
150:        z.push_back(' ');
151:        z.push_back(y.at(i));
152:        z.push_back(' ');
153:        z.push_back('2');
154:        z.push_back('\n');
155:        i++;
156:     }
157:
158:    return z;
159: }
```

PS5a: Ring Buffer and Guitar Hero Part A

Part one of program five. This part focused mainly on implementing the ring buffer which was essentially a queue. I coded enqueue, dequeue, and peek functions. Enqueue added an item to the end of the queue, dequeue removed the first item in the queue and returned it to the user, and peek returns the first item in the queue without removing it. Boost testing was used in this section to test that the queue was functioning as it should, and would be ready for part two of this assignment. This was my first time using a queue in C++, as the only other time I had used one was when I implemented one in C in Computing II. This was also the first time I implemented CPP Lint on any of my assignments. I formatted my code to comply with Google's standard coding style, and continued to do this for all assignments going forward. While I may not agree with everything in the style guide, I do admit that the code looks a lot cleaner because of it.



Makefile Mon Mar 06 16:09:09 2017 1

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=test.o RingBuffer.o
 4: P=ps5a
 5: BOOST= -lboost_unit_test_framework
 6: SFML= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 7: all: $(P)
 8: $(P):$(O)
 9:         $(C) -o $(P) $(O) $(BOOST) $(SFML)
10:
11: $(E).o:
12:         $(C) -c $<  -o $@
13:
14: clean:
15:         rm $(O) $(P)
```

test.cpp Mon Mar 06 19:01:24 2017 1

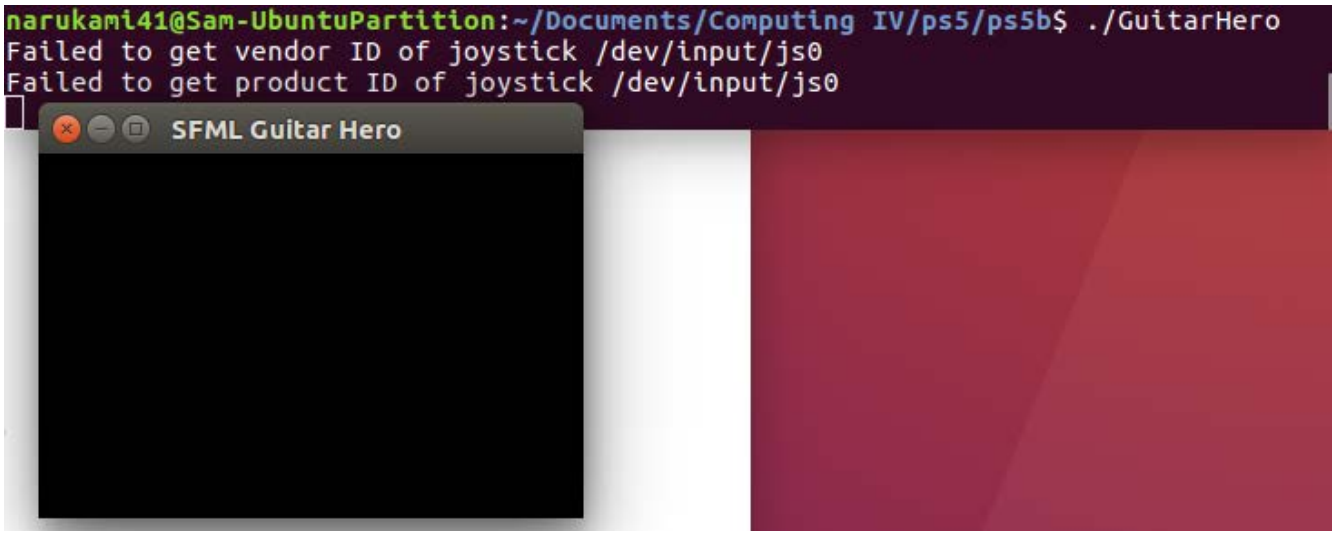
```
 1: /*Copyright 2017 Sam Pickell*/
 2: #define BOOST_TEST_DYN_LINK
 3: #define BOOST_TEST_MODULE Main
 4: #include <boost/test/unit_test.hpp>
 5: #include <string>
 6: #include <exception>
 7: #include "RingBuffer.hpp"
 8:
 9: BOOST_AUTO_TEST_CASE(RBcontructor) {
10:         // normal constructor
11:         BOOST_REQUIRE_NO_THROW(RingBuffer(100));
12:
13:         // this should fail
14:         BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
15:         BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
16: }
17:
18: BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
19:         RingBuffer rb(3);
20:
21:         rb.enqueue(2);
22:         rb.enqueue(1);
23:         rb.enqueue(0);
24:
25:         BOOST_REQUIRE(rb.dequeue() == 2);
26:         BOOST_REQUIRE(rb.dequeue() == 1);
27:         BOOST_REQUIRE(rb.dequeue() == 0);
28:
29:         BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
30: }
31:
32: BOOST_AUTO_TEST_CASE(RBpeek) {
33:         RingBuffer rb(3);
34:
35:         rb.enqueue(42);
36:         rb.enqueue(100);
37:         rb.enqueue(9001);
38:
39:         // Can't enqueue full buffer
40:         BOOST_REQUIRE_THROW(rb.enqueue(1), std::runtime_error);
41:
42:         // Check peek
43:         BOOST_REQUIRE(rb.peek() == 42);
44:
45:         BOOST_REQUIRE(rb.dequeue() == 42);
46:         BOOST_REQUIRE(rb.dequeue() == 100);
47:         BOOST_REQUIRE(rb.dequeue() == 9001);
48:
49:         BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
50:         // Can't peek into an empty buffer
51:         BOOST_REQUIRE_THROW(rb.peek(), std::runtime_error);
52: }
```

```
 1: /* Copyright 2017 Sam Pickell */
 2: #ifndef RINGBUFFER_HPP
 3: #define RINGBUFFER_HPP
 4:
 5: #include <stdint.h>
 6: #include <stdexcept>
 7: #include <iostream>
 8: #include <vector>
 9:
10: class RingBuffer {
11:  public:
12:   RingBuffer();
13:   explicit RingBuffer(int u_capacity);
14:   ~RingBuffer();
15:
16:   int size() { return my_size; }
17:   int get_capacity() { return capacity; }
18:   bool isEmpty();
19:   bool isFull();
20:   void enqueue(int16_t x);
21:   int16_t dequeue();
22:   int16_t peek();
23:
24:  private:
25:   int my_size, capacity;
26:   std::vector<int16_t> data;
27: };
28: #endif
```

```
 1: /*Copyright 2017 Sam Pickell*/
 2: #include "RingBuffer.hpp"
 3:
 4: RingBuffer::RingBuffer() {
 5:   my_size = 0;
 6:   capacity = 1;
 7: }
 8:
 9: RingBuffer::RingBuffer(int u_capacity) {
10:   my_size = 0;
11:
12:   if (u_capacity < 1) {
13:        throw std::invalid_argument(
14:             "RB constructor: capacity must be greater than zero.");
15:     } else {
16:        capacity = u_capacity;
17:     }
18: }
19:
20: RingBuffer::~RingBuffer() {}
21:
22: bool RingBuffer::isEmpty() {
23: return (my_size == 0);
24: }
25:
26: bool RingBuffer::isFull() {
27: return (my_size == capacity);
28: }
29:
30: void RingBuffer::enqueue(int16_t x) {
31:   if (this->isFull()) {
32:        throw std::runtime_error("enqueue: can't enqueue to a full ring.");
33:     } else {
34:       data.push_back(x);
35:       my_size++;
36:     }
37: }
38:
39: int16_t RingBuffer::dequeue() {
40:   int16_t temp;
41:   if (this->isEmpty()) {
42:        throw std::runtime_error("dequeue: nothing to dequeue, empty.");
43:     } else {
44:       temp = data.front();
45:       data.erase(data.begin());
46:       my_size--;
47:     }
48:   return temp;
49: }
50:
51: int16_t RingBuffer::peek() {
52:   if (this->isEmpty()) {
53:        throw std::runtime_error("peek: nothing to see, empty.");
54:     }
55:   return data.front();
56: }
```

PS5b: Ring Buffer and Guitar Hero Part B

Part two of program five. This part of the assignment uses Ring Buffer in tandem with a Guitar String object. Guitar String emulates a guitar string by playing different notes based on what key the user presses. This is done by making parallel vectors of sixteen bit integers, sf::soundbuffers, and sf::sounds. After the Guitar String is plucked and ticked, the sixteen bit integer that represents frequency is then loaded into a sound buffer, and the buffer is then loaded into a sound. This is the first program involving SFML that involved audio, and showcases how versatile SFML is.



Makefile Mon Mar 20 11:27:49 2017 1

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=GuitarHero.o RingBuffer.o GuitarString.o
 4: P=GuitarHero
 5: SFML= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 6: all: $(P)
 7: $(P):$(O)
 8:         $(C) -o $(P) $(O) $(SFML)
 9:
10: $(E).o:
11:         $(C) -c $<  -o $@
12:
13: clean:
14:         rm $(O) $(P)
```

```
    1: /*Copyright Sam Pickell 2017*/
    2: #include <limits.h>
    3: #include <cmath>
    4: #include <iostream>
    5: #include <string>
    6: #include <exception>
    7: #include <stdexcept>
    8: #include <vector>
    9:
   10: #include "GuitarString.hpp"
   11:
   12: #define CONCERT_A 440.0
   13: #define SAMPLES_PER_SEC 44100
   14:
   15: std::vector<sf::Int16> makeSamplesFromString(GuitarString& gs);
   16:
   17: int main() {
   18:   sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Guitar Hero");
   19:   sf::Event event;
   20:   std::vector< std::vector<sf::Int16> > my_samples;
   21:   std::vector<sf::SoundBuffer*> my_buffers;
   22:   std::vector<sf::Sound> my_sounds;
   23:
   24:
   25:   for (int i = 0; i < 37; i++) {
   26:       // Step 1
   27:       double freq = CONCERT_A * pow(2, (static_cast<double>(i - 24.0) / 12.0
));
   28:       GuitarString gs = GuitarString(freq);
   29:       my_samples.push_back(makeSamplesFromString(gs));
   30:
   31:       // Step 2
   32:       sf::SoundBuffer* my_buf = new sf::SoundBuffer;
   33:       if (!my_buf->loadFromSamples(
   34:         &(my_samples.at(i))[0], my_samples.at(i).size(), 2, SAMPLES_PER_SEC)
) {
   35:           throw std::runtime_error(
   36:         "sf::SoundBuffer: failed to load from samples.");
   37:         }
   38:       my_buffers.push_back(my_buf);
   39:
   40:       // Step 3
   41:       sf::Sound my_sound;
   42:       my_sound.setBuffer(*my_buf);
   43:       my_sounds.push_back(my_sound);
   44:     }
   45:
   46:   while (window.isOpen()) {
   47:     while (window.pollEvent(event)) {
   48:       switch (event.type) {
   49:       case sf::Event::Closed:
   50:         window.close();
   51:         break;
   52:
   53:       case sf::Event::KeyPressed:
   54:         switch (event.key.code) {
   55:         case sf::Keyboard::Q:
   56:           my_sounds.at(0).play();
   57:           break;
   58:           case sf::Keyboard::Num2:
   59:           my_sounds.at(1).play();
```

```
 60:             break;
 61:             case sf::Keyboard::W:
 62:             my_sounds.at(2).play();
 63:             break;
 64:             case sf::Keyboard::E:
 65:             my_sounds.at(3).play();
 66:             break;
 67:             case sf::Keyboard::Num4:
 68:             my_sounds.at(4).play();
 69:             break;
 70:             case sf::Keyboard::R:
 71:             my_sounds.at(5).play();
 72:             break;
 73:             case sf::Keyboard::Num5:
 74:             my_sounds.at(6).play();
 75:             break;
 76:             case sf::Keyboard::T:
 77:             my_sounds.at(7).play();
 78:             break;
 79:             case sf::Keyboard::Y:
 80:             my_sounds.at(8).play();
 81:             break;
 82:             case sf::Keyboard::Num7:
 83:             my_sounds.at(9).play();
 84:             break;
 85:             case sf::Keyboard::U:
 86:             my_sounds.at(10).play();
 87:             break;
 88:             case sf::Keyboard::Num8:
 89:             my_sounds.at(11).play();
 90:             break;
 91:             case sf::Keyboard::I:
 92:             my_sounds.at(12).play();
 93:             break;
 94:             case sf::Keyboard::Num9:
 95:             my_sounds.at(13).play();
 96:             break;
 97:             case sf::Keyboard::O:
 98:             my_sounds.at(14).play();
 99:             break;
100:             case sf::Keyboard::P:
101:             my_sounds.at(15).play();
102:             break;
103:             case sf::Keyboard::Dash:
104:             my_sounds.at(16).play();
105:             break;
106:             case sf::Keyboard::LBracket:
107:             my_sounds.at(17).play();
108:             break;
109:             case sf::Keyboard::Equal:
110:             my_sounds.at(18).play();
111:             break;
112:             case sf::Keyboard::Z:
113:             my_sounds.at(19).play();
114:             break;
115:             case sf::Keyboard::X:
116:             my_sounds.at(20).play();
117:             break;
118:             case sf::Keyboard::D:
119:             my_sounds.at(21).play();
120:             break;
```

```
121:            case sf::Keyboard::C:
122:            my_sounds.at(22).play();
123:            break;
124:            case sf::Keyboard::F:
125:            my_sounds.at(23).play();
126:            break;
127:            case sf::Keyboard::V:
128:            my_sounds.at(24).play();
129:            break;
130:            case sf::Keyboard::G:
131:            my_sounds.at(25).play();
132:            break;
133:            case sf::Keyboard::B:
134:            my_sounds.at(26).play();
135:            break;
136:            case sf::Keyboard::N:
137:            my_sounds.at(27).play();
138:            break;
139:            case sf::Keyboard::J:
140:            my_sounds.at(28).play();
141:            break;
142:            case sf::Keyboard::M:
143:            my_sounds.at(29).play();
144:            break;
145:            case sf::Keyboard::K:
146:            my_sounds.at(30).play();
147:            break;
148:            case sf::Keyboard::Comma:
149:            my_sounds.at(31).play();
150:            break;
151:            case sf::Keyboard::Period:
152:            my_sounds.at(32).play();
153:            break;
154:            case sf::Keyboard::SemiColon:
155:            my_sounds.at(33).play();
156:            break;
157:            case sf::Keyboard::Slash:
158:            my_sounds.at(34).play();
159:            break;
160:            case sf::Keyboard::Quote:
161:            my_sounds.at(35).play();
162:            break;
163:            case sf::Keyboard::Space:
164:            my_sounds.at(36).play();
165:            break;
166:          default:
167:            break;
168:          }
169:
170:        default:
171:          break;
172:        }
173:
174:        window.clear();
175:        window.display();
176:      }
177:    }
178:
179:    // Clean up after the vector of pointers
180:    for (unsigned int i = 0; i < my_buffers.size(); i++) {
181:        delete my_buffers.at(i);
```

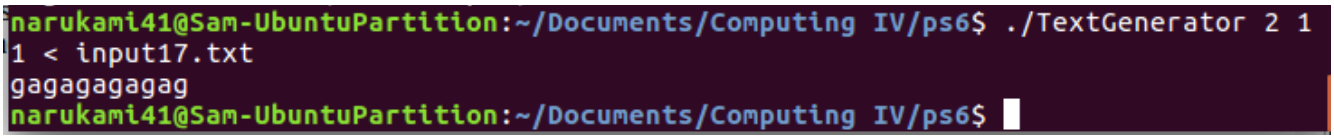
```
182:     }
183:   return 0;
184: }
185:
186: std::vector<sf::Int16> makeSamplesFromString(GuitarString& gs) {
187:   std::vector<sf::Int16> samples;
188:
189:   gs.pluck();
190:   int duration = 8;
191:   int i;
192:   for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
193:     gs.tic();
194:     samples.push_back(gs.sample());
195:   }
196:
197:   return samples;
198: }
```

```
 1: /*Copyright Sam Pickell 2017*/
 2:
 3: #ifndef GUITARSTRING_HPP
 4: #define GUITARSTRING_HPP
 5:
 6: #include <SFML/Graphics.hpp>
 7: #include <SFML/System.hpp>
 8: #include <SFML/Window.hpp>
 9: #include <SFML/Audio.hpp>
10: #include <vector>
11: #include "RingBuffer.hpp"
12:
13: class GuitarString {
14:  public:
15:   GuitarString();
16:   explicit GuitarString(double frequency);
17:   explicit GuitarString(std::vector<sf::Int16> init);
18:   ~GuitarString();
19:   void pluck();
20:   void tic();
21:   sf::Int16 sample();
22:   int time();
23:
24:  private:
25:   RingBuffer* data;
26:   int tic_tracker;
27: };
28: #endif
```

```
 1: /*Copyright Sam Pickell 2017*/
 2: #include "GuitarString.hpp"
 3: #include <cmath>
 4: #include <cstdlib>
 5: #include <vector>
 6:
 7: GuitarString::GuitarString() {
 8: }
 9:
10: GuitarString::GuitarString(double frequency) {
11:   int N = ceil(44100/frequency);
12:   data = new RingBuffer(N);
13:   for (int i = 0; i < N; i++) {
14:       data->enqueue(0);
15:     }
16: }
17:
18: GuitarString::GuitarString(std::vector<sf::Int16> init) {
19:   data = new RingBuffer(init.size());
20:   for (unsigned int i = 0; i < init.size(); i++) {
21:       data->enqueue(init.at(i));
22:     }
23: }
24:
25: GuitarString::~GuitarString() {
26:   delete data;
27: }
28:
29: void GuitarString::pluck() {
30:   unsigned int r = 123;
31:   for (int i = 0; i < data->size(); i++) {
32:       int16_t random_var = (int16_t)(rand_r(&r) % 0xffff);
33:       data->dequeue();
34:       data->enqueue(random_var);
35:     }
36: }
37:
38: void GuitarString::tic() {
39:   int16_t KS_update = data->peek();
40:   data->dequeue();
41:   KS_update = ((KS_update + data->peek()) / 2) * .996;
42:   data->enqueue(KS_update);
43:
44:   tic_tracker++;
45: }
46:
47: sf::Int16 GuitarString::sample() {
48:   return data->peek();
49: }
50:
51: int GuitarString::time() {
52:   return tic_tracker;
53: }
```

PS6: Markov Model of Natural Language

Program six tasked me to predict the trajectory of letters based on a string in a file. This program required me to use the Standard Template Library's map to properly keep track of the k-grams. The user inputs a number "n" and the input string is broken up into "n" k-grams. The final output is determined by the frequencies of characters that follow one another, and based on those probabilities, the output changes. It was interesting to learn that similar algorithms are used in sentence auto generation.



Makefile Mon Apr 03 12:49:28 2017 1

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O= MarkovModel.o TextGenerator.o
 4: P=TextGenerator
 5: BOOST= -lboost_unit_test_framework
 6: all: $(P)
 7: $(P):$(O)
 8:         $(C) -o $(P) $(O) $(BOOST)
 9:
10: $(E).o:
11:         $(C) -c $<  -o $@
12:
13: clean:
14:         rm $(O) $(P)
```

```
 1: // Copyright 2017 Sam Pickell
 2: #include <cstdlib>
 3: #include <string>
 4: #include "MarkovModel.hpp"
 5:
 6: int main(int argc, char* argv[]) {
 7:   int k, T;
 8:   std::string my_string;
 9:
10:   k = atoi(argv[1]);
11:   T = atoi(argv[2]);
12:   std::cin >> my_string;
13:
14:   MarkovModel MM(my_string, k);
15:
16:   std::cout << MM.gen(my_string.substr(0, k), T) << std::endl;
17:   return 0;
18: }
```

MarkovModel.hpp Mon Apr 03 16:23:51 2017 1

```
 1: // Copyright 2017 Sam Pickell
 2: #ifndef MARKOVMODEL_HPP
 3: #define MARKOVMODEL_HPP
 4:
 5: #include <string>
 6: #include <map>
 7: #include <iostream>
 8: #include <stdexcept>
 9:
10: class MarkovModel {
11:  public:
12:   MarkovModel(std::string text, int k);
13:   ~MarkovModel();
14:   int order() {return private_order;}
15:   int freq(std::string kgram);
16:   int freq(std::string kgram, char c);
17:   char randk(std::string kgram);
18:   std::string gen(std::string kgram, int T);
19:
20:   friend std::ostream& operator<< (std::ostream &out, MarkovModel &mm);
21:
22:  private:
23:   unsigned int private_order;
24:   std::map <std::string, int> kgrams;
25:   std::string alphabet;
26: };
27: #endif
```

```
 1: // Copyright 2017 Sam Pickell
 2: #include <cstdlib>
 3: #include <vector>
 4: #include <ctime>
 5: #include <map>
 6: #include <string>
 7: #include "MarkovModel.hpp"
 8:
 9: MarkovModel::MarkovModel(std::string text, int k) {
10:   unsigned int i;
11:   std::string kgram;
12:
13:   for (i = 0; i < text.size() - k; i++) {
14:       kgram = text.substr(i, k);
15:       kgrams[kgram]++;
16:       // look for a new character. If found, add to our alphabet string
17:       std::size_t my_char = alphabet.find(text.at(i));
18:       if (my_char == std::string::npos) {
19:            alphabet.push_back(text.at(i));
20:       }
21:       kgram.push_back(text.at(i+k));
22:       kgrams[kgram]++;
23:   }
24:   for (unsigned int j = i; j <= text.size() - 1; j++) {
25:       int l;
26:       kgram = text.substr(j, text.size() - j);
27:       l = k - (text.size() - j);
28:       kgram.append(text.substr(0, l));
29:       kgrams[kgram]++;
30:       // look for a new character. If found, add to our alphabet string
31:       std::size_t my_char = alphabet.find(text.at(i));
32:       if (my_char == std::string::npos) {
33:            alphabet.push_back(text.at(i));
34:         }
35:       kgram.push_back(text.at(l));
36:       kgrams[kgram]++;
37:     }
38:
39:   private_order = k;
40: }
41:
42: MarkovModel::~MarkovModel() {
43: }
44:
45: int MarkovModel::freq(std::string kgram) {
46:   if (kgram.size() != private_order) {
47:        throw std::runtime_error(
48:             "freq: kgram is not of length k!");
49:     } else {
50:     return kgrams[kgram];
51:     }
52: }
53:
54: int MarkovModel::freq(std::string kgram, char c) {
55:   if (kgram.size() != private_order) {
56:        throw std::runtime_error(
57:             "freq: kgram is not of length k!");
58:     } else {
59:       if (private_order == 0) {
60:           std::string adv_c;
61:           adv_c.push_back(c);
```

```
   62:             return kgrams[adv_c];
   63:           } else {
   64:             kgram.push_back(c);
   65:             return kgrams[kgram];
   66:           }
   67:       }
   68: }
   69:
   70: char MarkovModel::randk(std::string kgram) {
   71:   if (kgram.size() != private_order) {
   72:       throw std::runtime_error(
   73:             "randk: kgram is not of length k!");
   74:   } else if (!kgrams[kgram]) {
   75:       throw std::runtime_error(
   76:             "randk: kgram does not exist!");
   77:   } else {
   78:     std::vector<char> my_vector;
   79:     srand(time(NULL));
   80:     unsigned int random_var = (rand() % freq(kgram)); // NOLINT
   81:     for (unsigned int i = 0; i < alphabet.size(); i++) {
   82:         int my_freq = freq(kgram, alphabet.at(i));
   83:         if (my_freq >= 1) {
   84:             for (int j = 0; j < my_freq; j++) {
   85:               my_vector.push_back(alphabet.at(i));
   86:               }
   87:           }
   88:       }
   89:     return my_vector.at(random_var);
   90:     }
   91: }
   92:
   93: std::string MarkovModel::gen(std::string kgram, int T) {
   94:   std::string ret_string = kgram;
   95:   int i = 0;
   96:   while (ret_string.size() < (static_cast<unsigned int>(T))) {
   97:       std::string temp;
   98:       char c;
   99:       temp = ret_string.substr(i, private_order);
  100:       c = randk(temp);
  101:       ret_string.push_back(c);
  102:       i++;
  103:     }
  104:   return ret_string;
  105: }
  106:
  107: std::ostream& operator<< (std::ostream &out, MarkovModel &mm) {
  108:   for (std::map<std::string, int>::iterator p = mm.kgrams.begin();
  109:       p != mm.kgrams.end(); p++) {
  110:       out << p->first << '-' << p->second << std::endl;
  111:     }
  112:   out << "Order: " << mm.private_order << std::endl;
  113:   out << "Alphabet: " << mm.alphabet << std::endl;
  114:   return out;
  115: }
```

PS7a: Kronos Intouch Parsing Part A

Part one of program seven. The program introduced me to regular expressions and large text file parsing. The program would read a large log file from a medical device and report start and end time, and what the boot time was. Learning how to use regular expressions was particularly useful as this adds more variety to what can be received as input.

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

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

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

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

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

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

Makefile Sun Apr 09 16:24:40 2017 1

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=main.o
 4: P=ps7a
 5: BOOST= -lboost_regex -lboost_date_time
 6: all: $(P)
 7: $(P):$(O)
 8:         $(C) -o $(P) $(O) $(BOOST)
 9:
10: $(E).o:
11:         $(C) -c $<  -o $@
12:
13: clean:
14:         rm $(O) $(P)
```

```
    1: // Copyright 2017 Sam Pickell
    2: #include <boost/regex.hpp>
    3: #include <cstdlib>
    4: #include <iostream>
    5: #include <string>
    6: #include <fstream>
    7: #include "boost/date_time/gregorian/gregorian.hpp"
    8: #include "boost/date_time/posix_time/posix_time.hpp"
    9:
   10: int main(int argc, char* argv[]) {
   11:   int line_number = 1;
   12:   boost::regex start_up("[0-9]{4}-[0-9]{2}-[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[0-9
]"
   13: "{2}:\\s[()]log[.]c[.]166[)]\\sserver\\sstarted\\s");
   14:   boost::regex success("[0-9]{4}-[0-9]{2}-[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[0-9]
{2}"
   15: "[.][0-9]{3}:INFO:oejs[.]AbstractConnector:Started\\sSelectChannelConnector@
[0-"
   16: "9]{1}[.][0-9]{1}[.][0-9]{1}[.][0-9]{1}:[0-9]{4}");
   17:   boost::posix_time::time_duration diff;
   18:   std::string time_start;
   19:   std::string time_end;
   20:   std::ifstream fin;
   21:   std::ofstream fout;
   22:   std::string s, t;
   23:   std::string input_file = argv[1];
   24:   std::string output_file = input_file;
   25:   output_file.append(".rpt");
   26:
   27:   // Open files
   28:   fin.open(input_file.c_str());
   29:   if (fin.fail()) {
   30:       std::cout << "Failed to open file" << std::endl;
   31:       exit(1);
   32:     }
   33:   fout.open(output_file.c_str());
   34:   if (fout.fail()) {
   35:       std::cout << "Failed to open output file" << std::endl;
   36:       exit(1);
   37:     }
   38:   // End opening
   39:
   40:   std::getline(fin, s);
   41:   while (!fin.eof()) {
   42:       if (boost::regex_match(t, start_up) && boost::regex_match(s, success))
 {
   43:           // success
   44:           fout << line_number << "(" << input_file << "): " << s.substr(0, 1
9)
   45:                << " Boot Completed" << std::endl;
   46:           // boot time
   47:           time_end = s.substr(0, 19);
   48:           boost::posix_time::ptime start =
   49:             boost::posix_time::time_from_string(time_start);
   50:           boost::posix_time::ptime end =
   51:             boost::posix_time::time_from_string(time_end);
   52:           diff = end - start;
   53:           fout << "Boot Time: " << diff.total_milliseconds() << "ms"
   54:                << std::endl;
   55:           fout << std::endl;
   56:           t = "empty";
```

```
   57:         } else if (boost::regex_match(t, start_up) &&
   58:                boost::regex_match(s, start_up)) {
   59:           // failure
   60:           fout << "**** Incomplete boot ****" << std::endl;
   61:           fout << std::endl;
   62:           t = "empty";
   63:         }
   64:
   65:         if (boost::regex_match(s, start_up)) {
   66:           fout << "=== Device boot ===" << std::endl;
   67:           fout << line_number << "(" << input_file << "): " << s.substr(0, 1
9)
   68:                << " Boot Start" << std::endl;
   69:           time_start = s.substr(0, 19);
   70:           t = s;
   71:         }
   72:       std::getline(fin, s);
   73:       line_number++;
   74:     }
   75:
   76:   fin.close();
   77:   fout.close();
   78:
   79:   return 0;
   80: }
```

PS7b: Kronos Intouch Parsing Part B

Part two of program seven. This part of the program expanded on the previous one, where in addition to the start and end times I also had the program report on the different services and if they started up or not and how much time that took. Finally, I checked the logs for update information and reported on that too. Below is one section of one of the logs.

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

Services
      Logging
            Start: 435386(device1_intouch.log)
            Completed: 435387(device1_intouch.log)
            Elapsed Time: 346 ms
      DatabaseInitialize
            Start: 435388(device1_intouch.log)
            Completed: 435412(device1_intouch.log)
            Elapsed Time: 33139 ms
      MessagingService
            Start: 435413(device1_intouch.log)
            Completed: 435415(device1_intouch.log)
            Elapsed Time: 8663 ms
      HealthMonitorService
            Start: 435538(device1_intouch.log)
            Completed: 435539(device1_intouch.log)
            Elapsed Time: 239 ms
      Persistence
            Start: 435540(device1_intouch.log)
            Completed: 435541(device1_intouch.log)
            Elapsed Time: 23466 ms
      ConfigurationService
            Start: 435542(device1_intouch.log)
            Completed: 435543(device1_intouch.log)
            Elapsed Time: 0 ms
      LandingPadService
            Start: 435554(device1_intouch.log)
            Completed: 435555(device1_intouch.log)
            Elapsed Time: 2 ms
      PortConfigurationService
            Start: 435544(device1_intouch.log)
            Completed: 435545(device1_intouch.log)
            Elapsed Time: 100 ms
```

```
CacheService
      Start: 435556(device1_intouch.log)
      Completed: 435557(device1_intouch.log)
      Elapsed Time: 1413 ms
ThemingService
      Start: 435552(device1_intouch.log)
      Completed: 435553(device1_intouch.log)
      Elapsed Time: 1 ms
StagingService
      Start: 435558(device1_intouch.log)
      Completed: 435576(device1_intouch.log)
      Elapsed Time: 8816 ms
DeviceIOService
      Start: 435546(device1_intouch.log)
      Completed: 435551(device1_intouch.log)
      Elapsed Time: 48 ms
BellService
      Start: 435634(device1_intouch.log)
      Completed: 435637(device1_intouch.log)
      Elapsed Time: 207 ms
GateService
      Start: 435640(device1_intouch.log)
      Completed: 435643(device1_intouch.log)
      Elapsed Time: 5 ms
ReaderDataService
      Start: 435621(device1_intouch.log)
      Completed: 435622(device1_intouch.log)
      Elapsed Time: 4 ms
BiometricService
      Start: 435577(device1_intouch.log)
      Completed: 435620(device1_intouch.log)
      Elapsed Time: 269 ms
StateManager
      Start: 435625(device1_intouch.log)
      Completed: 435626(device1_intouch.log)
      Elapsed Time: 2468 ms
OfflineSmartviewService
      Start: 435638(device1_intouch.log)
      Completed: 435639(device1_intouch.log)
      Elapsed Time: 16 ms
AVFeedbackService
      Start: 435627(device1_intouch.log)
      Completed: 435633(device1_intouch.log)
      Elapsed Time: 123 ms
DatabaseThreads
      Start: 435743(device1_intouch.log)
      Completed: 435754(device1_intouch.log)
      Elapsed Time: 3596 ms
SoftLoadService
      Start: 435745(device1_intouch.log)
      Completed: 435752(device1_intouch.log)
      Elapsed Time: 3319 ms
```

```
      WATCHDOG
            Start: 435746(device1_intouch.log)
            Completed: 435750(device1_intouch.log)
            Elapsed Time: 526 ms
      ProtocolService
            Start: 435744(device1_intouch.log)
            Completed: Not completed(device1_intouch.log)
            Elapsed Time:
      DiagnosticsService
            Start: 435748(device1_intouch.log)
            Completed: 435749(device1_intouch.log)
            Elapsed Time: 218 ms
      *** Services not successfully started: ProtocolService
=== Softload ===
436276(device1_intouch.log) : Mar 25 19:17:55 Softload Start
      Original version ==> 2.0.0-uix.64
      New version ==> 2.0.2-146
      Elapsed time (sec) ==> 622
436427(device1_intouch.log) : Mar 25 19:28:17 Softload Completed
--------------------------------------------------------------------------------------------------------------------------
```

Makefile Sun Apr 16 17:40:55 2017 1

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=main.o
 4: P=ps7b
 5: BOOST= -lboost_regex -lboost_date_time
 6: all: $(P)
 7: $(P):$(O)
 8:         $(C) -o $(P) $(O) $(BOOST)
 9:
10: $(E).o:
11:         $(C) -c $<  -o $@
12:
13: clean:
14:         rm $(O) $(P)
```

```
    1: // Copyright 2017 Sam Pickell
    2: #include <boost/regex.hpp>
    3: #include <cstdlib>
    4: #include <iostream>
    5: #include <string>
    6: #include <fstream>
    7: #include <vector>
    8: #include <sstream>
    9: #include "boost/date_time/gregorian/gregorian.hpp"
   10: #include "boost/date_time/posix_time/posix_time.hpp"
   11:
   12: void populate_vectors(std::vector<std::string>& a, std::vector<std::string>&
 b,
   13:                       std::vector<std::string>& c, std::vector<std::string>&
 d);
   14: void print_stats(std::vector<std::string>& a, std::vector<std::string>& b,
   15:                      std::vector<std::string>& c, std::vector<std::string>&
d,
   16:                      std::ofstream& fout, std::ifstream& fin, std::string na
me);
   17:
   18: int main(int argc, char* argv[]) {
   19:   int line_number = 1;
   20:
   21:   // regex
   22:   boost::regex start_up("[0-9]{4}-[0-9]{2}-[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[0-9
]"
   23: "{2}:\\s[(]log[.]c[.]166[)]\\sserver\\sstarted\\s");
   24:   boost::regex success("[0-9]{4}-[0-9]{2}-[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[0-9]
{2}"
   25: "[.][0-9]{3}:INFO:oejs[.]AbstractConnector:Started\\sSelectChannelConnector@
[0-"
   26: "9]{1}[.][0-9]{1}[.][0-9]{1}[.][0-9]{1}:[0-9]{4}");
   27:   boost::regex services("Starting\\sService[.].*");
   28:   boost::regex service_success("Service\\sstarted\\ssuccessfully[.].*");
   29:   boost::regex soft_start("[A-Z]{1}[a-z]{2}\\s[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[
0-9"
   30: "]{2}\\s.*Install\\sstarted\\s");
   31:   boost::regex soft_orig("[A-Z]{1}[a-z]{2}\\s[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[0
-9"
   32: "]{2}\\s[(]none[)]\\s/stagingarea/scripts/install-rollback[.]sh:\\sintouch-a
ppl"
   33: "ication-base-.*[.]armv6jel_vfp[.]rpm\\swas\\spreviously\\sinstalled[,]\\sad
din"
   34: "g\\srpm\\sto\\srollback\\slist");
   35:   boost::regex soft_end("[A-Z]{1}[a-z]{2}\\s[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[0-
9"
   36: "]{2}\\s[(]none[)]\\s/stagingarea/scripts/install-rollback[.]sh:\\sExitValue
\\s"
   37: "from\\sinstall\\scommand\\s:\\s0");
   38:   boost::regex soft_new_ver("[A-Z]{1}[a-z]{2}\\s[0-9]{2}\\s[0-9]{2}:[0-9]{2}
:[0"
   39: "-9]{2}\\s[(]none[)]\\s/stagingarea/scripts/install-rollback[.]sh:\\sProcess
ing"
   40: "\\s43\\sof\\s45\\sintouch-platform-base-.*[.]armv6jel_vfp[.]rpm ...");
   41:
   42:   // vectors
   43:   std::vector<std::string> service_name;
   44:   std::vector<std::string> service_start;
   45:   std::vector<std::string> service_end;
   46:   std::vector<std::string> service_elapsed_time;
```

```
   47:
   48:   // time
   49:   boost::posix_time::time_duration diff;
   50:   std::string time_start;
   51:   std::string time_end;
   52:   std::string s_total_time;
   53:
   54:   std::ifstream fin;
   55:   std::ofstream fout;
   56:
   57:   std::string str_start = " ";
   58:   std::string str_end = " ";
   59:   std::string str_orig_ver = " ";
   60:   std::string str_new_ver = " ";
   61:   std::string str_elapsed = " ";
   62:   std::string s, t;
   63:   std::string service_1;
   64:   std::string input_file = argv[1];
   65:   std::string output_file = input_file;
   66:   output_file.append(".rpt");
   67:
   68:   int start_line;
   69:
   70:   // Open files
   71:   fin.open(input_file.c_str());
   72:   if (fin.fail()) {
   73:       std::cout << "Failed to open file" << std::endl;
   74:       exit(1);
   75:     }
   76:   fout.open(output_file.c_str());
   77:   if (fout.fail()) {
   78:       std::cout << "Failed to open output file" << std::endl;
   79:       exit(1);
   80:     }
   81:   // End opening
   82:
   83:   std::getline(fin, s);
   84:   while (!fin.eof()) {
   85:      // start of service startup
   86:
   87:     if (boost::regex_match(t, start_up) && boost::regex_match(s, services))
{
   88:         std::stringstream out;
   89:
   90:         for (unsigned int i = 0; i < service_name.size(); i++) {
   91:             std::string finder = service_name.at(i);
   92:             std::size_t pos = s.find(finder);
   93:             if (pos != std::string::npos) {
   94:                 service_1 = finder;
   95:                 out << line_number;
   96:                 service_start.at(i) = out.str();
   97:                 break;
   98:               }
   99:           }
  100:       }
  101:
  102:     if (boost::regex_match(t, start_up) &&
  103:         boost::regex_match(s, service_success)) {
  104:         std::stringstream out;
  105:         unsigned int i = 0;
  106:
```

```
  107:             for (i = 0; i < service_name.size(); i++) {
  108:                 std::string finder = service_name.at(i);
  109:                 std::size_t pos = s.find(finder);
  110:                 // if (service_1 == service_name.at(i))
  111:                 if (pos != std::string::npos &&
  112:                     (service_start.at(i) != "Not started")) {
  113:                     out << line_number;
  114:                     service_end.at(i) = out.str();
  115:
  116:                     // grab elapsed time
  117:                     std::size_t my_start = 0;
  118:                     std::size_t my_end = 0;
  119:                     std::string ela_time;
  120:
  121:                     my_start = s.find("(");
  122:                     if (my_start != std::string::npos) {
  123:                         ++my_start;
  124:                         my_end = s.find(")");
  125:                         if (my_end != std::string::npos) {
  126:                             ela_time = s.substr(my_start, my_end - my_start)
;
  127:                           }
  128:                       }
  129:                     service_elapsed_time.at(i) = ela_time;
  130:                     break;
  131:                   }
  132:               }
  133:       }
  134:      // end of service startup
  135:
  136:     // start of software upgrades
  137:
  138:     if (boost::regex_match(s, soft_start)) {
  139:         str_start = s;
  140:         start_line = line_number;
  141:       }
  142:     if (boost::regex_match(s, soft_end)) {
  143:         str_end = s;
  144:       }
  145:     if (boost::regex_match(s, soft_orig)) {
  146:         str_orig_ver = s;
  147:       }
  148:     if (boost::regex_match(s, soft_new_ver)) {
  149:         str_new_ver = s;
  150:       }
  151:
  152:     if (str_start != " " && str_end != " " &&
  153:         str_orig_ver != " " && str_new_ver != " ") {
  154:         // get elapsed time vars
  155:         std::string my_time_start = str_start.substr(0, 15);
  156:         std::string my_time_end = str_end.substr(0, 15);
  157:
  158:         int w, x, y;
  159:
  160:         w = atoi((my_time_start.substr(7, 2)).c_str());
  161:         x = atoi((my_time_start.substr(10, 2)).c_str());
  162:         y = atoi((my_time_start.substr(13, 2)).c_str());
  163:
  164:         boost::posix_time::time_duration time_to_start(w, x, y, 0);
  165:
  166:         w = atoi((my_time_end.substr(7, 2)).c_str());
```

```
  167:         x = atoi((my_time_end.substr(10, 2)).c_str());
  168:         y = atoi((my_time_end.substr(13, 2)).c_str());
  169:
  170:         boost::posix_time::time_duration time_to_end(w, x, y, 0);
  171:         boost::posix_time::time_duration my_diff;
  172:         my_diff = time_to_end - time_to_start;
  173:
  174:         // clean up orig_ver and new_ver
  175:         // grab orig
  176:         std::size_t my_start = 0;
  177:         std::size_t my_end = 0;
  178:         std::string temp;
  179:
  180:         my_start = str_orig_ver.find("base-");
  181:         if (my_start != std::string::npos) {
  182:           my_start+=5;
  183:           my_end = str_orig_ver.find("armv6");
  184:           if (my_end != std::string::npos) {
  185:             temp = str_orig_ver.substr(my_start, (my_end - my_start) - 1);
  186:           }
  187:         }
  188:         str_orig_ver = temp;
  189:
  190:         // now grab new
  191:         my_start = 0;
  192:         my_end = 0;
  193:
  194:         my_start = str_new_ver.find("base-");
  195:         if (my_start != std::string::npos) {
  196:           my_start+=5;
  197:           my_end = str_new_ver.find("armv6");
  198:           if (my_end != std::string::npos) {
  199:             temp = str_new_ver.substr(my_start, (my_end - my_start) - 1);
  200:           }
  201:         }
  202:         str_new_ver = temp;
  203:
  204:         // print out
  205:         fout << "=== Softload ===" << std::endl;
  206:         fout << start_line << "(" << input_file << ") : "
  207:              << str_start.substr(0, 15) << " Softload Start" << std::endl;
  208:         fout << "\tOriginal version ==> " << str_orig_ver << std::endl;
  209:         fout << "\tNew version ==> " << str_new_ver << std::endl;
  210:         fout << "\tElapsed time (sec) ==> " << my_diff.total_seconds()
  211:              << std::endl;
  212:         fout << line_number << "(" << input_file << ") : "
  213:              << str_end.substr(0, 15) << " Softload Completed" << std::endl;
  214:         str_start = " ";
  215:         str_end = " ";
  216:         str_orig_ver = " ";
  217:         str_new_ver = " ";
  218:       }
  219:
  220:     // end of software upgrades
  221:
  222:       if (boost::regex_match(t, start_up) && boost::regex_match(s, success))
 {
  223:           // success
  224:           fout << line_number << "(" << input_file << "): " << s.substr(0, 1
9)
  225:                << " Boot Completed" << std::endl;
```

```
  226:           // boot time
  227:           time_end = s.substr(0, 19);
  228:           boost::posix_time::ptime start =
  229:             boost::posix_time::time_from_string(time_start);
  230:           boost::posix_time::ptime end =
  231:             boost::posix_time::time_from_string(time_end);
  232:           diff = end - start;
  233:           fout << "\tBoot Time: " << diff.total_milliseconds() << "ms"
  234:                << std::endl;
  235:           fout << std::endl;
  236:           t = "empty";
  237:
  238:           // service printout
  239:           print_stats(service_name, service_start, service_end,
  240:                       service_elapsed_time, fout, fin, input_file);
  241:
  242:         } else if (boost::regex_match(t, start_up) &&
  243:                boost::regex_match(s, start_up)) {
  244:           // failure
  245:           fout << "**** Incomplete boot ****" << std::endl;
  246:           fout << std::endl;
  247:           t = "empty";
  248:
  249:           // service printout
  250:           print_stats(service_name, service_start, service_end,
  251:                       service_elapsed_time, fout, fin, input_file);
  252:         }
  253:
  254:         if (boost::regex_match(s, start_up)) {
  255:           fout << "=== Device boot ===" << std::endl;
  256:           fout << line_number << "(" << input_file << "): " << s.substr(0, 1
9)
  257:                << " Boot Start" << std::endl;
  258:           time_start = s.substr(0, 19);
  259:           t = s;
  260:
  261:           // service startup
  262:           populate_vectors(service_name, service_start, service_end,
  263:                    service_elapsed_time);
  264:         }
  265:       std::getline(fin, s);
  266:       line_number++;
  267:     }
  268:
  269:   fin.close();
  270:   fout.close();
  271:
  272:   return 0;
  273: }
  274:
  275: void populate_vectors(std::vector<std::string>& a, std::vector<std::string>&
 b,
  276:                       std::vector<std::string>& c, std::vector<std::string>&
d) {
  277:   a.push_back(" Logging");
  278:   b.push_back("Not started");
  279:   c.push_back("Not completed");
  280:   d.push_back("");
  281:
  282:   a.push_back(" DatabaseInitialize");
  283:   b.push_back("Not started");
```

```
284:   c.push_back("Not completed");
285:   d.push_back("");
286:
287:   a.push_back(" MessagingService");
288:   b.push_back("Not started");
289:   c.push_back("Not completed");
290:   d.push_back("");
291:
292:   a.push_back(" HealthMonitorService");
293:   b.push_back("Not started");
294:   c.push_back("Not completed");
295:   d.push_back("");
296:
297:   a.push_back(" Persistence");
298:   b.push_back("Not started");
299:   c.push_back("Not completed");
300:   d.push_back("");
301:
302:   a.push_back(" ConfigurationService");
303:   b.push_back("Not started");
304:   c.push_back("Not completed");
305:   d.push_back("");
306:
307:   a.push_back(" LandingPadService");
308:   b.push_back("Not started");
309:   c.push_back("Not completed");
310:   d.push_back("");
311:
312:   a.push_back(" PortConfigurationService");
313:   b.push_back("Not started");
314:   c.push_back("Not completed");
315:   d.push_back("");
316:
317:   a.push_back(" CacheService");
318:   b.push_back("Not started");
319:   c.push_back("Not completed");
320:   d.push_back("");
321:
322:   a.push_back(" ThemingService");
323:   b.push_back("Not started");
324:   c.push_back("Not completed");
325:   d.push_back("");
326:
327:   a.push_back(" StagingService");
328:   b.push_back("Not started");
329:   c.push_back("Not completed");
330:   d.push_back("");
331:
332:   a.push_back(" DeviceIOService");
333:   b.push_back("Not started");
334:   c.push_back("Not completed");
335:   d.push_back("");
336:
337:   a.push_back(" BellService");
338:   b.push_back("Not started");
339:   c.push_back("Not completed");
340:   d.push_back("");
341:
342:   a.push_back(" GateService");
343:   b.push_back("Not started");
344:   c.push_back("Not completed");
```

```
  345:   d.push_back("");
  346:
  347:   a.push_back(" ReaderDataService");
  348:   b.push_back("Not started");
  349:   c.push_back("Not completed");
  350:   d.push_back("");
  351:
  352:   a.push_back(" BiometricService");
  353:   b.push_back("Not started");
  354:   c.push_back("Not completed");
  355:   d.push_back("");
  356:
  357:   a.push_back(" StateManager");
  358:   b.push_back("Not started");
  359:   c.push_back("Not completed");
  360:   d.push_back("");
  361:
  362:   a.push_back(" OfflineSmartviewService");
  363:   b.push_back("Not started");
  364:   c.push_back("Not completed");
  365:   d.push_back("");
  366:
  367:   a.push_back(" AVFeedbackService");
  368:   b.push_back("Not started");
  369:   c.push_back("Not completed");
  370:   d.push_back("");
  371:
  372:   a.push_back(" DatabaseThreads");
  373:   b.push_back("Not started");
  374:   c.push_back("Not completed");
  375:   d.push_back("");
  376:
  377:   a.push_back(" SoftLoadService");
  378:   b.push_back("Not started");
  379:   c.push_back("Not completed");
  380:   d.push_back("");
  381:
  382:   a.push_back(" WATCHDOG");
  383:   b.push_back("Not started");
  384:   c.push_back("Not completed");
  385:   d.push_back("");
  386:
  387:   a.push_back(" ProtocolService");
  388:   b.push_back("Not started");
  389:   c.push_back("Not completed");
  390:   d.push_back("");
  391:
  392:   a.push_back(" DiagnosticsService");
  393:   b.push_back("Not started");
  394:   c.push_back("Not completed");
  395:   d.push_back("");
  396: }
  397:
  398: void print_stats(std::vector<std::string>& a, std::vector<std::string>& b,
  399:                 std::vector<std::string>& c, std::vector<std::string>& d,
  400:                 std::ofstream& fout, std::ifstream& fin, std::string name)
{
  401:   std::vector<std::string> boot_failures;
  402:   unsigned int my_size = a.size();
  403:
  404:   fout << "Services" << std::endl;
```

```
405:
406:    for (unsigned int i = 0; i < my_size; i++) {
407:        a.at(i).erase(0, 1);
408:        fout << "\t" << a.at(i) << std::endl;
409:        fout << "\t\tStart: " << b.at(i)
410:             << "(" << name << ")" << std::endl;
411:        fout << "\t\tCompleted: " << c.at(i)
412:             << "(" << name << ")" <<  std::endl;
413:        if (c.at(i) == "Not completed") {
414:            boot_failures.push_back(a.at(i));
415:          }
416:        fout << "\t\tElapsed Time: " << d.at(i) << std::endl;
417:      }
418:    if (boot_failures.size() > 0) {
419:        fout << "\t*** Services not successfully started: ";
420:        for (unsigned int i = 0; i < boot_failures.size(); i++) {
421:            if (i != (boot_failures.size() - 1)) {
422:                fout << boot_failures.at(i) << ", ";
423:              } else {
424:                fout << boot_failures.at(i) << std::endl;
425:              }
426:          }
427:      }
428:    a.clear();
429:    b.clear();
430:    c.clear();
431:    d.clear();
432: }
```