12/12/2018

Michael Treacy

COMP IV - 201: Project Portfolio

Fall 2018

Contents:

Introduction:

This portfolio contains my completed assignments for Professor Rykalova's COMP IV – 201 course taken in the Fall of 2018. For each assignment there exists a discussion section followed by the passed in source code. Counting each part of the multipart PS projects as its own assignment, there are a total of 11 assignments covered. I separated part a and part b of the multipart projects and included each of their source codes because I made small changes between versions. It should also be mentioned that I was only able to add line numbers to my source code by first opening my projects in Visual Studio, moving the code over to Notepad++, adding line numbers in Notepad++, and then copying and pasting here.

1. Assignment: PS0: Hello World with SFML

2. General discussion and what was accomplished:

The focus of this assignment was to get my build environment set up and experiment a bit with SFML. After running the SFML "did I install everything correctly" code, I extended the demo code so that an image sprite was drawn (Figure 1), the sprite moved and responded to keystrokes, and it tilted itself to the right (Figure 2).

3. One or more key algorithms, data structures, or OO designs central to assignment:

Making a sprite move in response to an arrow key being pressed involved creating if-statements of the following format:

```
46      if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)){
47    z = 1;
48    sprite.move(-3, 0);
49      }
```

As it can be seen, I simply made it so that when an arrow key was pressed the sprite.move() function would be called. The -3 means the sprite will move to the left by 3. The z was a variable that caused the code between lines 65 and 76 to not be executed. Those lines, which can be viewed in the source code included at the end of this assignment discussion, moved the sprite on a set path.

4. What was learned:

This project introduced me to some of the SFML libraries we would be using throughout the semester. It was nice that the documentation was fairly extensive, and most functions were self-explanatory given their name.

5. Evidence of success:

Figure 1

Figure 2

6. Problems:

I completed all aspects of the assignment and did not encounter any serious problems. Making the sprite stop moving on its own when a key was pressed was a bit tricky, but all I had to do was create a flag-like variable that would change when an arrow key was hit. The value of this variable, z, would then determine whether the code for automatic movement would be run.

7. Source code:

main.cpp

```
/*******************************
Author: Michael Treacy
Date: 9/10/18
(Other information in readme)
 ******************************/

#include <SFML/Graphics.hpp>

int main()
{
  // Coordinate values and flag for whether key pressed
  int x = 350, y = 350;
  int z = 0;

  // Create window and set framerate
  sf::RenderWindow window(sf::VideoMode(500, 500), "Skull Kid and the Moon");
  window.setFramerateLimit(60);

  // Create circle
  sf::CircleShape shape(100.f);
  shape.setFillColor(sf::Color::White);

  // Load sprite
  sf::Texture texture;
  if(!texture.loadFromFile("sprite.png")){
    return EXIT_FAILURE;
  }
  sf::Sprite sprite(texture);

  // Set its scale and initial position
  sprite.setScale(sf::Vector2f(0.2f, 0.2f));
```

```
sprite.setPosition(x, y);

// Start loop
while(window.isOpen()){
    sf::Event event;
    while (window.pollEvent(event)){
    if(event.type == sf::Event::Closed)
      window.close();
    }

    window.clear();

    // Move sprite if direction key pressed. z is changed so that
    // sprite stops moving automatically
    if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)){
  z = 1;
  sprite.move(-3, 0);
    }
    if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
  z = 1;
  sprite.move(3, 0);
    }
    if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up)){
  z = 1;
  sprite.move(0, -3);
    }
    if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down)){
  z = 1;
  sprite.move(0, 3);
    }

    // Sprite will move to center until a key is pressed
    // If final pos. met, sprite rotated 15 degrees
    if(z == 0){
  if(x > 190){
    x -= 2;
  }
  if(x <= 190 && y > 190){
    y -= 2;
  }
  if(x <= 190 && y <= 190){
    sprite.setRotation(15);
  }
  sprite.setPosition(x, y);
    }

    window.draw(shape);
    window.draw(sprite);
    window.display();
}

return 0;
}
```

1. Assignment: PS1: Recursive Graphics (Pythagoras tree)

2. General discussion and what was accomplished:

This assignment involved creating a recursive function that would ultimately create a Pythagoras tree based on the user's desired initial length of the square and depth of recursion.

3. One or more key algorithms, data structures, or OO designs central to assignment:

The key algorithm would probably be the recursive one I built in the file PTree.cpp. Basically, I created a function called pTree that would create a square off two passed in points, (x0, y0) and (x1, y1), and then use the two new top points that were created to find the third point of the right triangle that would fit nicely on top of the newly created square. Finally, pTree would be called once with the top left point of the square and the third triangle point and once with the third triangle point and the top right point. This would continue until N, or the number of iterations, was down to 0.

I left fairly detailed comments in my code pertaining to the exact geometric math I was doing for each step. I'll explain it in some detail here. The main idea was that I would calculate the slope of the line between the passed in base points, but keep the rise and run in separate variables so that I could flip them and multiply the nominator by -1. This is seen in the code excerpt below:

```
// Slope calculation
rise = y1 - y0;
run = x1 - x0;
// Flip and multiply by -1
temp = rise;
rise = run * -1;
run = temp;
```

I did this so that I could get the perpendicular slope. With this, I was able to obtain the unknown coordinates. After making the newSquare with the points I passed into the function as well as the new points just calculated, I went about getting the third point of the triangle placed above the top just calculated points. This was done by finding the midpoint between (x3, y3) and (x2, y2) and then subtracting from this a vector that was half the length of the line between these two points, but rotated 90 degrees. The rotation looked as follows:

```
// Rotate it 90 degrees
temp = v;
v = w * -1;
w = temp;
```

Once x4 and y4 were calculated from the above subtraction, I called pTree with x3, y3, x4 and y4 for the left side and again with x4, y4, x2 and y2 for the right side. N-1 and the window was also included as well.

4. What was learned:

This project really forced me tackle the geometry of the square and triangle before I did any coding. When I first started, I was just trying random different things, which, to be honest, got me nowhere. It wasn't until I sat down and took out a notepad that I finally was able to plan out my recursive function. Once the math was worked out, the function proved to be straightforward. In all, this assignment really taught me the benefits of coming up with a good plan for your code.

5. Evidence of success:

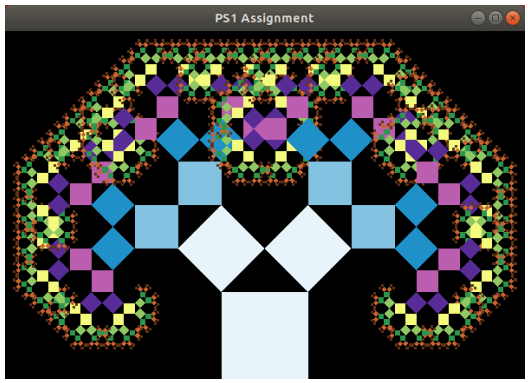

Figure 3

6. Problems:

I did not try to change the degrees of the tree because I was not really sure how I would go about manipulating them. For the 90-45-45 triangle, the third point was always in the middle of the base points and the length from the base pretty straightforward. I haven't been able to wrap my head around changing how I would calculate the triangle point. It should be mentioned that I did do the extra credit of adding color to the tree (Figure 3). The color changes based on the inputted depth of recursion.

7. Source code:

Makefile

```
CC = g++
CFLAGS = -Wall -Werror -std=c++0x -pedantic
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
OBJS = main.o PTree.o

all: coloredTree

coloredTree: $(OBJS)
    $(CC) $(OBJS) -o coloredTree $(LFLAGS)

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

clean:
    rm *.o coloredTree *~
```

main.cpp

```
#include <iostream>
#include <string>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include "PTree.hpp"

int main(int argc, char* argv[])
{
  // Take double and integer arguments separated by a space
  double L = std::stoi(argv[1]);
  int N = std::stoi(argv[2]);

  sf::RenderWindow window(sf::VideoMode(6 * L, 4 * L), "PS1 Assignment");
```

```
  window.setFramerateLimit(60);

  // Calculate position of trunkSquare and create it
  double x0, y0, x1, y1, x2, y2, x3, y3;
  x0 = 3 * L - 0.5 * L;
  y0 = 4 * L;
  x1 = 3 * L + 0.5 * L;
  y1 = 4 * L;
  x2 = 3 * L + 0.5 * L;
  y2 = 3 * L;
  x3 = 3 * L - 0.5 * L;
  y3 = 3 * L;
  PTree trunkSquare(x0, y0, x1, y1, x2, y2, x3, y3, N);

  // Calculate third point of triangle sitting on square
  double x4, y4;
  x4 = x0 + 0.5 * L;
  y4 = y3 - 0.5 * L;

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

    window.draw(trunkSquare);
    // Call recursive func. pTree for left square and right square
    pTree(window, N, x3, y3, x4, y4);
    pTree(window, N, x4, y4, x2, y2);

    window.display();
  }

  return 0;
}
```

PTree.hpp

```
#pragma once
#ifndef PTREE_HPP
#define PTREE_HPP

#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>

class PTree : public sf::Drawable
{
public:
  PTree(double x0, double y0, double x1, double y1, double x2, double y2, double
x3, double y3, int i);

  void draw(sf::RenderTarget& target, sf::RenderStates states) const;
private:
  sf::ConvexShape convexS;
};

```

```
void pTree(sf::RenderWindow &window, int N, double x0, double y0, double x1, double y1);

#endif
```

PTree.cpp

```
#include <iostream>
#include <cmath>
#include "PTree.hpp"

// Constructor that creates convex shape from inputted points
PTree::PTree(double x0, double y0, double x1, double y1, double x2, double y2, double x3, double y3, int i){
  convexS.setPointCount(4);
  convexS.setPoint(0, sf::Vector2f(x0, y0));
  convexS.setPoint(1, sf::Vector2f(x1, y1));
  convexS.setPoint(2, sf::Vector2f(x2, y2));
  convexS.setPoint(3, sf::Vector2f(x3, y3));
  if(i == 0){
    convexS.setFillColor(sf::Color::Green);
  }
  else{
    convexS.setFillColor(sf::Color(i * 100, i * 50, i * 25));
  }
}

// PTree's draw function
void PTree::draw(sf::RenderTarget& target, sf::RenderStates states) const {
  target.draw(convexS);
}

// Recursive function
void pTree(sf::RenderWindow &window, int N, double x0, double y0, double x1, double y1){
  if(N == 0){
    return;
  }
  else{
    // Calculate slope of line perpendicular to line between known points
    // by getting the slope of line between known points, flipping its
    // nominator and denominator, and then multiplying it by -1.
    // Use perpendicular slope to find the two unknown points by working
    // off two knowns
    double x2, y2, x3, y3;
    double rise, run, temp;

    // Slope calculation
    rise = y1 - y0;
    run = x1 - x0;
    // Flip and multiply by -1
    temp = rise;
    rise = run * -1;
    run = temp;
    // Use perpendicular slope to get unknown coordinates
    x2 = x1 + run;
    y2 = y1 + rise;
    x3 = x0 + run;
    y3 = y0 + rise;
```

```

    // Make new square with the 4 points
    PTree newSquare(x0, y0, x1, y1, x2, y2, x3, y3, N);
    window.draw(newSquare);

    // Calculate third point by finding midpoint of (x3, y3) and (x2, y2),
    // dividing that by 2, and then subtracting a rotated vector that's
    // half the length of the line between the points
    double x4, y4;
    double m, n, v, w;

    // Find midpoint
    m = x3 + x2;
    m *= 0.5;
    n = y3 + y2;
    n *= 0.5;
    // Find vector that's half the length of the line between the points
    v = x2 - x3;
    v *= 0.5;
    w = y2 - y3;
    w *= 0.5;
    // Rotate it 90 degrees
    temp = v;
    v = w * -1;
    w = temp;
    // Subtract (m, n) and (v, w)
    x4 = m - v;
    y4 = n - w;

    // Call pTree recursive func. for left new square and right new square
    pTree(window, N-1, x3, y3, x4, y4);
    pTree(window, N-1, x4, y4, x2, y2);
  }
}
```

1. Assignment: PS2a: Linear Feedback Shift Register: Linear Feedback Shift Register and Unit Testing

2. General discussion and what was accomplished:

I basically implemented the LFSR class by writing the code for the constructor, step and generate function, and overloaded <<stream insertion operator. I then became acquainted with the Boost library by writing 7 extra tests. To be fair, one was simply an implementation of the example output given in the assignment handout sheet.

3. One or more key algorithms, data structures, or OO designs central to assignment:

When it came to shifting the bits to the left, I simply used a for-loop that moved the next bit into the position of the current bit. As for the generate() function, I implemented it exactly as the assignment sheet instructed. In other words, I multiplied the variable by 2 and added the new bit returned by step(). This was done k times. The variable started at 0. As for the representation I used for the register bits, I decided to store it as a string. This seemed logical considering I was passing in a string seed and could just set registerValue = seed. I could also obtain the length easily, as well as the correct tap bit. I say the correct tap bit because leaving it as the passed in t would not work due to the string index starting from the left side and increasing to the right. The math I used to calculate the right tap bit was to take the

size, subtract 1, and then subtract from that the passed in t. All of this mentioned code can be viewed in the LFSR.hpp and LFSR.cpp files located at the end of this assignment section.

When it came to my additional tests, I created a total of 7 more. The first was an implementation of the output data included on the assignment sheet. The other 6 ranged from testing the edge cases to testing the generate() function with low k-values. More specifically, I made sure that a one-bit seed with a tap of 0 returned 0 for each step() and generate() call. Along with this, I tested the next smallest seed length of two bits with a tap of 0 and made sure the step() and generate() calls returned the right values. Stepping through things on paper and then making sure the values matched was straightforward. Here is an excerpt of the BOOST_AUTO_TEST_CASE(twoBitsTapAtZero) test:

```
LFSR l9("10", 0);
BOOST_REQUIRE(l9.step() == 1);
BOOST_REQUIRE(l9.step() == 1);
BOOST_REQUIRE(l9.step() == 0);
BOOST_REQUIRE(l9.step() == 1);
BOOST_REQUIRE(l9.step() == 1);
BOOST_REQUIRE(l9.step() == 0);

LFSR l10("10", 0);
BOOST_REQUIRE(l10.generate(6) == 54);
```

Most of my tests took on this basic form with changes to the seed, tap, number of times step() and generate() were called, and name of my LFSR object. With that said, I also tested a 32-bit string at the arbitrary tap of 10, a 32-bit with tap at 0 and 30, as well as the generate() function with the low k-values of 0, 1, and 2. I tested the overloaded<<stream insertion operator with a main.cpp file, but I did not include that file here due to it not being a required part of the project.

4. What was learned:

The main thing I learned from this assignment was how to use some of the features of the Boost library. This was a great thing to learn considering I hear that a lot of internships start out with interns simply writing tests.

5. Evidence of success:

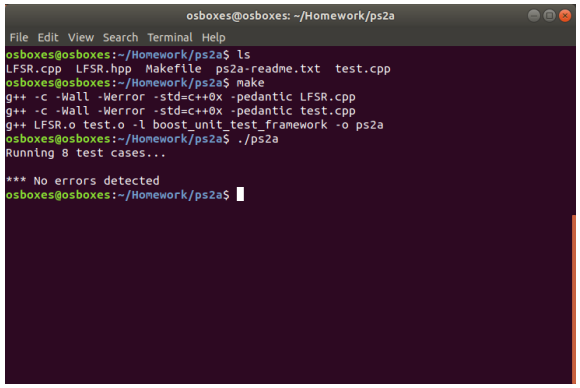

Figure 4

6. Problems:

I did not encounter any serious problems while writing this program. All parts were completed without

too much trouble (Figure 4). The tests were just a bit time consuming to think of and write out.

7. Source code:

Makefile

```
all: ps2a

ps2a: LFSR.o test.o
    g++ LFSR.o test.o -l boost_unit_test_framework -o ps2a
LFSR.o: LFSR.cpp LFSR.hpp
    g++ -c -Wall -Werror -std=c++0x -pedantic LFSR.cpp
test.o: test.cpp
    g++ -c -Wall -Werror -std=c++0x -pedantic test.cpp
clean:
    rm *.o ps2a *~
```

test.cpp

```
#include <iostream>
#include <string>

#include "LFSR.hpp"

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

// The example test used to check if calling step() 8 times on
// a five-bit string would return the correct int each time.
// The return value of generate(8) is also tested
BOOST_AUTO_TEST_CASE(fiveBitsTapAtTwo) {

  LFSR l("00111", 2);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);

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

// Test for making sure the return values of step() and
// generate(5) are the same as the shown output on the
// assignment description sheet
BOOST_AUTO_TEST_CASE(elevenBitsTapAtEight) {

  LFSR l3("01101000010", 8);
  BOOST_REQUIRE(l3.step() == 1);
  BOOST_REQUIRE(l3.step() == 1);
  BOOST_REQUIRE(l3.step() == 0);
  BOOST_REQUIRE(l3.step() == 0);
  BOOST_REQUIRE(l3.step() == 1);
  BOOST_REQUIRE(l3.step() == 0);
```

```
041  BOOST_REQUIRE(l3.step() == 0);
042  BOOST_REQUIRE(l3.step() == 1);
043  BOOST_REQUIRE(l3.step() == 0);
044  BOOST_REQUIRE(l3.step() == 0);
045
046  LFSR l4("01101000010", 8);
047  BOOST_REQUIRE(l4.generate(5) == 25);
048  BOOST_REQUIRE(l4.generate(5) == 4);
049  BOOST_REQUIRE(l4.generate(5) == 30);
050  BOOST_REQUIRE(l4.generate(5) == 27);
051  BOOST_REQUIRE(l4.generate(5) == 18);
052  BOOST_REQUIRE(l4.generate(5) == 26);
053  BOOST_REQUIRE(l4.generate(5) == 28);
054  BOOST_REQUIRE(l4.generate(5) == 24);
055  BOOST_REQUIRE(l4.generate(5) == 23);
056  BOOST_REQUIRE(l4.generate(5) == 29);
057}
058
059// Test for checking that smallest possible seed string
060// of 0 or 1 will always have step() and generate()
061// return 0 (makes sense considering XORing same position)
062BOOST_AUTO_TEST_CASE(oneBitTapAtZero) {
063
064  LFSR l5("0", 0);
065  BOOST_REQUIRE(l5.step() == 0);
066  BOOST_REQUIRE(l5.step() == 0);
067  BOOST_REQUIRE(l5.step() == 0);
068  BOOST_REQUIRE(l5.step() == 0);
069
070  LFSR l6("0", 0);
071  BOOST_REQUIRE(l6.generate(4) == 0);
072
073  LFSR l7("1", 0);
074  BOOST_REQUIRE(l7.step() == 0);
075  BOOST_REQUIRE(l7.step() == 0);
076  BOOST_REQUIRE(l7.step() == 0);
077  BOOST_REQUIRE(l7.step() == 0);
078
079  LFSR l8("1", 0);
080  BOOST_REQUIRE(l8.generate(4) == 0);
081}
082
083// Test for checking that step() and generate(6) work
084// correctly for next smallest seed string
085BOOST_AUTO_TEST_CASE(twoBitsTapAtZero) {
086
087  LFSR l9("10", 0);
088  BOOST_REQUIRE(l9.step() == 1);
089  BOOST_REQUIRE(l9.step() == 1);
090  BOOST_REQUIRE(l9.step() == 0);
091  BOOST_REQUIRE(l9.step() == 1);
092  BOOST_REQUIRE(l9.step() == 1);
093  BOOST_REQUIRE(l9.step() == 0);
094
095  LFSR l10("10", 0);
096  BOOST_REQUIRE(l10.generate(6) == 54);
097}
098
099// Test for 32-bit string with arbitraty tap of ten
```

```
BOOST_AUTO_TEST_CASE(thirtyTwoBitsTapAtTen) {

  LFSR l11("10010101101001011100100011010011", 10);
  BOOST_REQUIRE(l11.step() == 1);
  BOOST_REQUIRE(l11.step() == 0);
  BOOST_REQUIRE(l11.step() == 0);
  BOOST_REQUIRE(l11.step() == 0);
  BOOST_REQUIRE(l11.step() == 1);
  BOOST_REQUIRE(l11.step() == 1);

  LFSR l12("10010101101001011100100011010011", 10);
  BOOST_REQUIRE(l12.generate(6) == 35);
}

// Test for 32-bit string with tap at edge zero
BOOST_AUTO_TEST_CASE(thirtyTwoBitsTapAtZero) {

  LFSR l13("10010001110010010110101101000100", 0);
  BOOST_REQUIRE(l13.step() == 1);
  BOOST_REQUIRE(l13.step() == 1);
  BOOST_REQUIRE(l13.step() == 1);
  BOOST_REQUIRE(l13.step() == 0);
  BOOST_REQUIRE(l13.step() == 0);

  LFSR l14("10010001110010010110101101000100", 0);
  BOOST_REQUIRE(l14.generate(5) == 28);
}

// Test for 32-bit string with tap at edge 30
BOOST_AUTO_TEST_CASE(thirtyTwoBitsTapAtThirty) {

  LFSR l15("10010001110010010110101101000100", 30);
  BOOST_REQUIRE(l15.step() == 1);
  BOOST_REQUIRE(l15.step() == 0);
  BOOST_REQUIRE(l15.step() == 1);
  BOOST_REQUIRE(l15.step() == 1);
  BOOST_REQUIRE(l15.step() == 0);

  LFSR l16("10010001110010010110101101000100", 30);
  BOOST_REQUIRE(l16.generate(5) == 22);
}

// Test for generate() function and whether it works
// correctly for calls with a smaller k
BOOST_AUTO_TEST_CASE(generateZeroOneAndTwo){

  LFSR l17("100110", 2);
  BOOST_REQUIRE(l17.generate(0) == 0);

  LFSR l18("100010", 2);
  BOOST_REQUIRE(l18.generate(0) == 0);

  LFSR l19("100110", 2);
  BOOST_REQUIRE(l19.generate(1) == 0);

  LFSR l20("100010", 2);
  BOOST_REQUIRE(l20.generate(1) == 1);

  LFSR l21("100110", 2);
```

```
  BOOST_REQUIRE(l21.generate(2) == 1);

  LFSR l22("100010", 2);
  BOOST_REQUIRE(l22.generate(2) == 3);
}
```

LFSR.hpp

```
#pragma once
#ifndef LFSR_HPP
#define LFSR_HPP

#include <iostream>
#include <string>

class LFSR {
public:
  LFSR(std::string seed, int t);

  int step();

  int generate(int k);

  friend std::ostream& operator<< (std::ostream &out, const LFSR &lfsr);
private:
  std::string registerValue;
  int size;
  int tap;
};

#endif
```

LFSR.cpp

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

// Constructor for LFSR class that takes initial seed and tap
LFSR::LFSR(std::string seed, int t){
  registerValue = seed;
  size = registerValue.length();
  tap = (size - 1) - t;
}

// Function that simulates one step of the LFSR and returns
// the new rightmost bit
int LFSR::step(){
  char tapBit, lastBit, newBit;
  tapBit = registerValue[tap];
  lastBit = registerValue[0];

  // Find newBit by XORing tapBit and lastBit
  if(tapBit == lastBit){
    newBit = '0';
  }
  else{
    newBit = '1';
  }

```

```
  // Shift bits to the left in string and add newBit to beginning
  for(int i = 0; i < size - 1; i++){
    registerValue[i] = registerValue[i + 1];
  }
  registerValue[size - 1] = newBit;

  // Return rightmost bit
  if(newBit == '1'){
    return 1;
  }
  else{
    return 0;
  }
}

int LFSR::generate(int k){
  int binaryRep = 0;

  // For each bit extracted, double the variable and add what
  // step() returns
  for(int i = 0; i < k; i++){
    binaryRep = binaryRep * 2 + step();
  }

  return binaryRep;
}

std::ostream& operator<< (std::ostream &out, const LFSR &lfsr){
  out << lfsr.registerValue;
  return out;
}
```

1. Assignment: PS2b: Linear Feedback Shift Register: Encoding/decoding images with LFSR

2. General discussion and what was accomplished:

This assignment involved making a program that could encrypt an image as well as decrypt it if it was run again with the encrypted image. This called for creating a sprite for the initial image passed in and then manipulating that image, creating a sprite for the manipulated version, and then outputting the original and the changed in two separate windows. The encryption process simply involved XORing each pixel's red, green, and blue colors with what was returned by the LFSR's generate function.

3. One or more key algorithms, data structures, or OO designs central to assignment:

The key design of ps2b was probably the manipulation of the pixels. As it can be seen from the code and comment below, the image was distorted by XORing the color of each pixel with a generated int. The two for-loops result in every pixel in the image being covered.

```
  // Manipulate image by XORing the color at each pixel with generated int
  int x, y;
  sf::Color pixel;
  for(x = 0; x < i; x++){
    for(y = 0; y < j; y++){
      pixel = image.getPixel(x, y);
      pixel.r ^= lfsr.generate(8);
      pixel.g ^= lfsr.generate(8);
```

```
      pixel.b ^= lfsr.generate(8);
      image.setPixel(x, y, pixel);
    }
  }
```

4. What was learned:

I did not know anything about encryption/decryption before I tackled this assignment. It always just seemed like a matter of randomly messing with pixels. Implementing the LFSR class and seeing how it could be used was interesting because it made me realize how one can piece back together a scrambled image (Figure 5 and Figure 6).

5. Evidence of success:

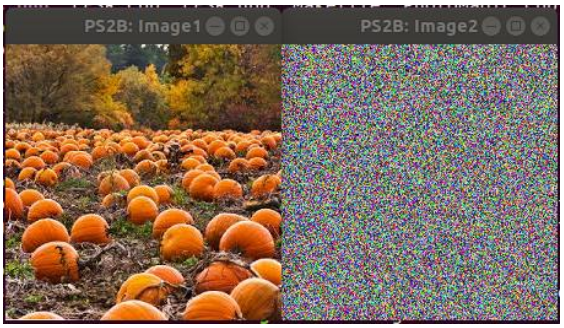

Figure 5

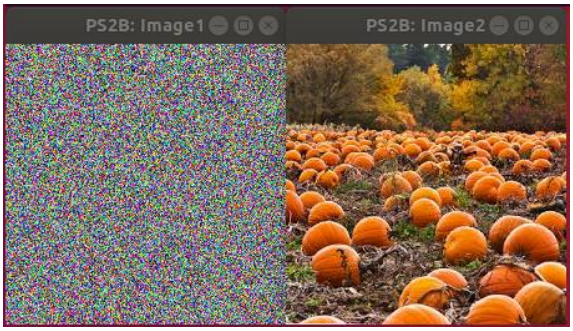

Figure 6

6. Problems:

I did not really encounter any serious problems. Deciding what I should set the k-value to when I called generate confused me a bit at first. I found out that 1-7 did not really encrypt the initial image enough, but 8 and up got the job done. I just chose 8 so that the program didn't take too long to go through the steps like the higher numbers did. As for extra credit, I did not attempt any added features due to a lack of time.

7. Source code:

Makefile

```
CC = g++
CFLAGS = -Wall -Werror -std=c++0x -pedantic
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
OBJS = PhotoMagic.o LFSR.o

all: PhotoMagic

PhotoMagic: $(OBJS)
    $(CC) $(OBJS) -o PhotoMagic $(LFLAGS)

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

clean:
    rm *.o PhotoMagic *~
```

PhotoMagic.cpp

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

int main(int argc, char* argv[])
{
  // Take command line arguments and create lfsr
  std::string inputFile = argv[1];
  std::string outputFile = argv[2];
  std::string registerSeed = argv[3];
  int tapPos = std::stoi(argv[4]);
  LFSR lfsr(registerSeed, tapPos);

  // Create source image to be manipulated
  sf::Image image;
  if(!image.loadFromFile(inputFile)){
    return -1;
  }

  // Find size of image
  sf::Vector2u pngSize = image.getSize();
  int i, j;
  i = pngSize.x;
  j = pngSize.y;

  // Create both windows for source image and manipulated image
  sf::RenderWindow window1(sf::VideoMode(i, j), "PS2B: Image1");
  sf::RenderWindow window2(sf::VideoMode(i, j), "PS2B: Image2");

  // Make texture and sprite for original before image is changed
  sf::Texture texture1;
  texture1.loadFromImage(image);
  sf::Sprite sprite1;
  sprite1.setTexture(texture1);

  // Manipulate image by XORing the color at each pixel with generated int
  int x, y;
  sf::Color pixel;
  for(x = 0; x < i; x++){
    for(y = 0; y < j; y++){
      pixel = image.getPixel(x, y);
      pixel.r ^= lfsr.generate(8);
      pixel.g ^= lfsr.generate(8);
      pixel.b ^= lfsr.generate(8);
      image.setPixel(x, y, pixel);
    }
  }

  // Create texture and sprite for changed image
  sf::Texture texture2;
  texture2.loadFromImage(image);
  sf::Sprite sprite2;
  sprite2.setTexture(texture2);

```

```
  // Event loop for displaying both windows
  while(window1.isOpen() && window2.isOpen()){
    sf::Event event;
    while(window1.pollEvent(event)){
      if(event.type == sf::Event::Closed){
    window1.close();
      }
    }
    while(window2.pollEvent(event)){
      if(event.type == sf::Event::Closed){
    window2.close();
      }
    }
    window1.clear();
    window1.draw(sprite1);
    window1.display();
    window2.clear();
    window2.draw(sprite2);
    window2.display();
  }

  // Write out the encrypted file to the output-file.png
  if(!image.saveToFile(outputFile)){
    return -1;
  }

  return 0;
}
```

LFSR.hpp

```
#pragma once
#ifndef LFSR_HPP
#define LFSR_HPP

#include <iostream>
#include <string>

class LFSR {
public:
  LFSR(std::string seed, int t);

  int step();

  int generate(int k);

  friend std::ostream& operator<< (std::ostream &out, const LFSR &lfsr);
private:
  std::string registerValue;
  int size;
  int tap;
};

#endif
```

LFSR.cpp

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

```
// Constructor for LFSR class that takes initial seed and tap
LFSR::LFSR(std::string seed, int t){
  registerValue = seed;
  size = registerValue.length();
  tap = (size - 1) - t;
}

// Function that simulates one step of the LFSR and returns
// the new rightmost bit
int LFSR::step(){
  char tapBit, lastBit, newBit;
  tapBit = registerValue[tap];
  lastBit = registerValue[0];

  // Find newBit by XORing tapBit and lastBit
  if(tapBit == lastBit){
    newBit = '0';
  }
  else{
    newBit = '1';
  }

  // Shift bits to the left in string and add newBit to beginning
  for(int i = 0; i < size - 1; i++){
    registerValue[i] = registerValue[i + 1];
  }
  registerValue[size - 1] = newBit;

  // Return rightmost bit
  if(newBit == '1'){
    return 1;
  }
  else{
    return 0;
  }
}

int LFSR::generate(int k){
  int binaryRep = 0;

  // For each bit extracted, double the varable and add what
  // step() returns
  for(int i = 0; i < k; i++){
    binaryRep = binaryRep * 2 + step();
  }

  return binaryRep;
}

std::ostream& operator<< (std::ostream &out, const LFSR &lfsr){
  out << lfsr.registerValue;
  return out;
}
```

1. Assignment: PS3a: N-Body Simulation: Loading universe files; body class; graphics

2. General discussion and what was accomplished:

This assignment involved creating a Body class that would store all the necessary data for each body, or particle, that is to be drawn. This was completed, along with the overloading of the input stream operator and drawing of each body at its initial position.

3. One or more key algorithms, data structures, or OO designs central to assignment:

I believe one of the main designs of this assignment involved overloading the input stream operator so that it loaded the parameter data for an object. This wasn't too bad to implement due to the fact that it was expected that the program would be given the planets.txt file to read from. Basically, I knew that whenever >> was used I would be reading from the file instead of the keyboard. With that said, the overloaded file simply involved using is >> and then the private Body variable to get it set. A look between lines 29 and 58 of Body.cpp reveal how this was done. The "is" refers to the istream variable.

The private virtual void method named draw was something I implemented in a previous assignment, so I just did the same thing for this one. Basically, I included the needed parameter list and put target.draw(particle) in the function body. This made it so that I could use window.draw(obj). I also loaded the universe from stdin by using std::cin >> R. N, the number of particles, is read just before since it is the first in the file.

```
// Read in the number of particles and radius of universe
double N, R;
std::cin >> N;
std::cin >> R;
```

Since my for-loop for filling my vector of Body ptrs goes up to one less than N (i starts at 0), my program supports an arbitrary number of body objects. Scaling works for arbitrary universe size and given SFML window size because they are both passed into the constructor for each Body. The math is done in the overloaded >>. As it can be seen below, I used unique_ptr's for my smart pointers. The vector holds unique_ptr's to Body objects which are given the universe radius and window size and then given the individual body data with the help of the overloaded input stream operator.

```
// Create vector of Body smart pointers
std::vector<std::unique_ptr<Body>> bodies;
// Could have used make_unique, but compiler was giving me a
// bit of trouble with it even though the syntax seemed
// completely ok
for(int i = 0; i < N; i++){
  bodies.push_back(std::unique_ptr<Body> (new Body(R, winR)));
  std::cin >> *bodies[i];
}
```

4. What was learned:

I feel like a learned a lot with this assignment. One surprising thing was how to read from a file with >> after it is passed in. I know I should have probably encountered this before, but it was kind of new to me.

5. Evidence of success:

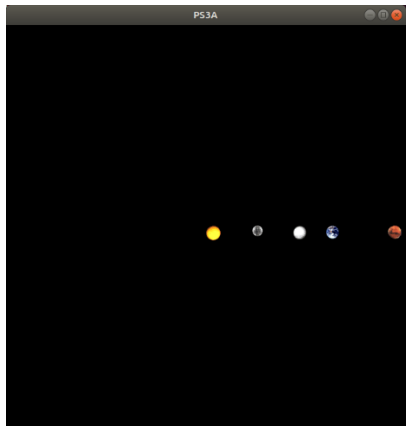

Figure 7

6. Problems:

I did not really encounter any serious problems. The make_unique compiler problems were strange, but I believe I read that make_unique is not standard until C++14. Anyway, I was able to implement that portion in an alternative way.

7. Source code:

Makefile

```
CC = g++
CFLAGS = -Wall -Werror -std=c++0x -pedantic
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
OBJS = main.o Body.o

all: NBody

NBody: $(OBJS)
    $(CC) $(OBJS) -o NBody $(LFLAGS)

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

clean:
    rm *.o NBody *~
```

main.cpp

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

int main(int argc, char* argv[])
{
  double winR = 300;
  double diameter = winR * 2;
  sf::RenderWindow window(sf::VideoMode(diameter, diameter), "PS3A");
```

```
15
16  // Read in the number of particles and radius of universe
17  double N, R;
18  std::cin >> N;
19  std::cin >> R;
20
21  // Create vector of Body smart pointers
22  std::vector<std::unique_ptr<Body>> bodies;
23  // Could have used make_unique, but compiler was giving me a
24  // bit of trouble with it even though the syntax seemed
25  // completely ok
26  for(int i = 0; i < N; i++){
27    bodies.push_back(std::unique_ptr<Body> (new Body(R, winR)));
28    std::cin >> *bodies[i];
29  }
30
31  // Event loop for displaying both windows
32  while(window.isOpen()){
33    sf::Event event;
34    while(window.pollEvent(event)){
35      if(event.type == sf::Event::Closed){
36    window.close();
37      }
38    }
39
40    window.clear();
41    // Draw each body in the vector of Body smart pointers
42    for(int i = 0; i < N; i++){
43      window.draw(*bodies[i]);
44    }
45
46    window.display();
47    }
48
49  return 0;
50}
```

Body.hpp

```
01#pragma once
02#ifndef BODY_HPP
03#define BODY_HPP
04
05#include <iostream>
06#include <string>
07#include <SFML/System.hpp>
08#include <SFML/Window.hpp>
09#include <SFML/Graphics.hpp>
10
11class Body : public sf::Drawable
12{
13public:
14  Body(double rUniverse, double rWindow);
15
16  friend std::istream& operator>>(std::istream& is, Body& b);
17
18private:
19  virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
20
```

```
  double R;
  double wR;
  double xpos;
  double ypos;
  double xvel;
  double yvel;
  double mass;
  std::string filename;
  sf::Texture texture;
  sf::Sprite particle;
};

#endif
```

Body.cpp

```
#include <iostream>
#include <string>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "Body.hpp"

// Default constructor that sets the radiuses of the universe and window
// to the correct values, as well as the private variables to 0. texure and
// particle are left alone because it is stated in the SFML documentation
// that Textures and Sprites are empty when their default constructor is
// called. strings are also empty when made with the default constructor so
// I did not touch filename
Body::Body(double rUniverse, double rWindow){
  R = rUniverse;
  wR = rWindow;
  xpos = 0;
  ypos = 0;
  xvel = 0;
  yvel = 0;
  mass = 0;
}

// Private virtual void method named draw
void Body::draw(sf::RenderTarget& target, sf::RenderStates states) const {
  target.draw(particle);
}

// Overrided input stream operator, which loads parameter data into object
std::istream& operator>>(std::istream& is, Body& b){
  // Store values (may adjust values like I did below depending on the
  // requirements of ps3b)
  is >> b.xpos;
  is >> b.ypos;
  is >> b.xvel;
  is >> b.yvel;
  is >> b.mass;
  is >> b.filename;

  // Create sprite
  sf::Image image;
  image.loadFromFile(b.filename);
  b.texture.loadFromImage(image);
```

```
b.particle.setTexture(b.texture);

// Place object in correct pixel position
double xPixel, yPixel;
xPixel = b.xpos / b.R;
xPixel *= b.wR;
yPixel = b.ypos / b.R;
yPixel *= b.wR;
// Add wR to each since origin in SFML is top left
xPixel += b.wR;
yPixel += b.wR;
b.particle.setPosition(sf::Vector2f(xPixel, yPixel));

return is;
}
```

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

2. General discussion and what was accomplished:

This assignment involved adding physics simulation and animation to my ps3a program. I basically added setters and getters to my Body class, implemented a function called calcNetForces which took the whole vector of smart pointers and calculated each particle's netForceX and netForceY, and then I created a step method that would calculate each particle's acceleration, new velocity, and new position. Lastly, I modified my setPosition function so that it could deal with the SFML y-axis and overloaded the output stream operator so that I could output the final state of each body.

3. One or more key algorithms, data structures, or OO designs central to assignment:

Since the calcNetForces function was probably the most difficult algorithm to write, I include a good chunk of it below:

```
for(int i = 0; i < sizeB; i++){
    (*bodies[i]).resetnetForceX();
    (*bodies[i]).resetnetForceY();
    for(int j = 0; j < sizeB; j++){
 if(j != i){
   changeX = (*bodies[j]).getxpos() - (*bodies[i]).getxpos();
   changeY = (*bodies[j]).getypos() - (*bodies[i]).getypos();
   r = sqrt(pow(changeX, 2.0) + pow(changeY, 2.0));
   F = G * (*bodies[j]).getmass() * (*bodies[i]).getmass();
   F /= pow(r, 2.0);
   (*bodies[i]).increasenetForceX((F * changeX) / r);
   (*bodies[i]).increasenetForceY((F * changeY) / r);
 }
    }
}
```

What I did was simply calculate G, which was done before these for-loops, reset the focused-on particle's x- and y- net force, and then add up the pairwise forces acting on the particle by going through the other particles. The syntax was a little strange due to the vector being passed in by reference, but I figured it was better to do this in a function than fill the main.cpp file with it.

4. What was learned:

I actually feel as though this program helped me to brush up on my physics skills. Along with this, I grew more comfortable with the way in which SFML's x- and y-coordinate system works. It is actually the case that I had the planets rotating backwards at one point.

5. Evidence of success:

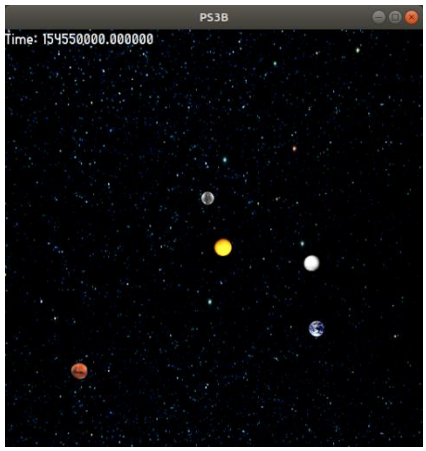

Figure 8

6. Problems:

I encountered some small problems along the way, but most of them just had to do with correcting the order in which I evaluated parts of an equation. This was always easy to fix. I did not create my own universe, but I did do the extra credit of displaying the elapsed time and playing a sound file (Figure 8). This was completed by following the examples and suggestions on the SFML website. Basically, I used the Music, Font, and Text SFML classes with a font and music file I found online. When it comes to the music I used, I guess the online creator wants to be credited. With that said, his name is Tanner Helland and this is his website: http://www.tannerhelland.com/music-directory/

7. Source code:

Makefile

```
CC = g++
CFLAGS = -Wall -Werror -std=c++0x -pedantic
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
OBJS = main.o Body.o

all: NBody

NBody: $(OBJS)
    $(CC) $(OBJS) -o NBody $(LFLAGS)

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

clean:
    rm *.o NBody *~
```

main.cpp

```
001#include <iostream>
002#include <string>
003#include <vector>
004#include <memory>
005#include <iomanip>
006#include <SFML/System.hpp>
007#include <SFML/Window.hpp>
008#include <SFML/Graphics.hpp>
009#include <SFML/Audio.hpp>
010#include "Body.hpp"
011
012int main(int argc, char* argv[])
013{
014  double winR = 256;
015  double diameter = winR * 2;
016  sf::RenderWindow window(sf::VideoMode(diameter, diameter), "PS3B");
017  window.setFramerateLimit(150);
018
019  // Setup background image by creating sprite with correct scale
020  sf::Texture texture;
021  if(!texture.loadFromFile("space.png")){
022    return EXIT_FAILURE;
023  }
024  sf::Sprite bSprite(texture);
025  sf::Vector2u texSize = texture.getSize();
026  bSprite.setScale( diameter / texSize.x, diameter / texSize.y);
027
028  // Add music and play it
029  sf::Music music;
030  if(!music.openFromFile("Deeper.ogg")){
031    return EXIT_FAILURE;
032  }
033  music.play();
034
035  // Read in T, changet, number of particles, and radius of universe
036  double T, changet, N, R;
037  T = std::stod(argv[1]);
038  changet = std::stod(argv[2]);
039  std::cin >> N;
040  std::cin >> R;
041
042  // Create vector of Body smart pointers
043  std::vector<std::unique_ptr<Body>> bodies;
044  // make_unique better, but we're using c++0x
045  for(int i = 0; i < N; i++){
046    bodies.push_back(std::unique_ptr<Body> (new Body(R, winR)));
047    std::cin >> *bodies[i];
048  }
049
050  // Set up text for displaying elapsed time
051  sf::Font font;
052  if(!font.loadFromFile("arial_narrow_7.ttf")){
053    return EXIT_FAILURE;
054  }
055  sf::Text text;
056  text.setFont(font);
057  text.setCharacterSize(18);
058
059  // Event loop for displaying window. t is start time
```

```
  double t = 0;
  text.setString("Time: " + std::to_string(t));
  while(window.isOpen()){
    sf::Event event;
    while(window.pollEvent(event)){
      if(event.type == sf::Event::Closed){
   window.close();
      }
    }

    window.clear();

    if(t != 0 && t <= T){
      // Step 1
      calcNetForces(bodies);
      // Step 2 and 3
      for(int i = 0; i < N; i++){
   (*bodies[i]).step(changet);
      }
      text.setString("Time: " + std::to_string(t));
    }

    // Close window if elapsed time goes beyond time limit
    if(t > T){
      window.close();
    }

    // Draw the background, each body in the vector of Body smart
    // pointers, and the elapsed time
    window.draw(bSprite);
    for(int i = 0; i < N; i++){
      window.draw(*bodies[i]);
    }
    window.draw(text);

    window.display();
    t += changet;
  }

  // Output final particle values
  std::cout << std::setw(13) << N << std::endl;
  std::cout << std::setw(13) << R << std::endl;
  for(int i = 0; i < N; i++){
    std::cout << *bodies[i] << std::endl;
  }

  return 0;
}
```

Body.hpp

```
#pragma once
#ifndef BODY_HPP
#define BODY_HPP

#include <iostream>
#include <string>
#include <vector>
#include <memory>
```

```
09#include <SFML/System.hpp>
10#include <SFML/Window.hpp>
11#include <SFML/Graphics.hpp>
12
13class Body : public sf::Drawable
14{
15public:
16  Body(double rUniverse, double rWindow);
17
18  double getxpos() { return xpos; }
19  void setxpos(double x) { xpos = x; }
20
21  double getypos() { return ypos; }
22  void setypos(double y) { ypos = y; }
23
24  double getxvel() { return xvel; }
25  void setxvel(double x) { xvel = x; }
26
27  double getyvel() { return yvel; }
28  void setyvel(double y) { yvel = y; }
29
30  double getmass() { return mass; }
31
32  void resetnetForceX() { netForceX = 0; }
33  void increasenetForceX(double x) { netForceX += x; }
34
35  void resetnetForceY() { netForceY = 0; }
36  void increasenetForceY(double y) { netForceY += y; }
37
38  void step(double changet);
39
40  void setPosition(void);
41
42  friend std::istream& operator>>(std::istream& is, Body& b);
43
44  friend std::ostream& operator<<(std::ostream& os, const Body& b);
45
46private:
47  virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
48
49  double R;
50  double wR;
51  double xpos;
52  double ypos;
53  double xvel;
54  double yvel;
55  double mass;
56  std::string filename;
57  sf::Image image;
58  sf::Texture texture;
59  sf::Sprite particle;
60
61  double netForceX;
62  double netForceY;
63};
64
65void calcNetForces(std::vector<std::unique_ptr<Body>>& bodies);
66
67#endif
```

Body.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <memory>
#include <cmath>
#include <iomanip>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "Body.hpp"

// Default constructor that sets the radiuses of the universe and window
// to the correct values, as well as the private variables to 0. image,
// texture, and particle are left alone because it is stated in the SFML
// documentation that Images, Textures and Sprites are empty when their
// default constructor is called. strings are also empty when made with
// the default constructor so I did not touch filename
Body::Body(double rUniverse, double rWindow){
  R = rUniverse;
  wR = rWindow;
  xpos = 0;
  ypos = 0;
  xvel = 0;
  yvel = 0;
  mass = 0;
  netForceX = 0;
  netForceY = 0;
}

// Private virtual void method named draw
void Body::draw(sf::RenderTarget& target, sf::RenderStates states) const {
  target.draw(particle);
}

// Overloaded stream input operator, which loads parameter data
// into object
std::istream& operator>>(std::istream& is, Body& b){
  is >> b.xpos;
  is >> b.ypos;
  is >> b.xvel;
  is >> b.yvel;
  is >> b.mass;
  is >> b.filename;

  // Create sprite particle
  b.image.loadFromFile(b.filename);
  b.texture.loadFromImage(b.image);
  b.particle.setTexture(b.texture);

  return is;
}

// Overloaded stream output operator so that final particle info
// can be given
std::ostream& operator<<(std::ostream& os, const Body& b){
  os << std::setw(13) << b.xpos;
```

```
057  os << std::setw(13) << b.ypos;
058  os << std::setw(13) << b.xvel;
059  os << std::setw(13) << b.yvel;
060  os << std::setw(13) << b.mass;
061  os << std::setw(13) << b.filename;
062
063  return os;
064}
065
066// Body-step 1 function that is passed the vector and calculates
067// the net force acting on each particle at a specific time
068void calcNetForces(std::vector<std::unique_ptr<Body>>& bodies){
069  double r, changeX, changeY, F, sizeB, G;
070  sizeB = bodies.size();
071  G = pow(10.0, -11.0);
072  G *= 6.67;
073
074  for(int i = 0; i < sizeB; i++){
075      (*bodies[i]).resetnetForceX();
076      (*bodies[i]).resetnetForceY();
077      for(int j = 0; j < sizeB; j++){
078   if(j != i){
079     changeX = (*bodies[j]).getxpos() - (*bodies[i]).getxpos();
080     changeY = (*bodies[j]).getypos() - (*bodies[i]).getypos();
081     r = sqrt(pow(changeX, 2.0) + pow(changeY, 2.0));
082     F = G * (*bodies[j]).getmass() * (*bodies[i]).getmass();
083     F /= pow(r, 2.0);
084     (*bodies[i]).increasenetForceX((F * changeX) / r);
085     (*bodies[i]).increasenetForceY((F * changeY) / r);
086   }
087      }
088  }
089}
090
091// Body-step 2 function that is passed changet, calculates the
092// acceleration, new velocity, and new position, and then calls
093// setPosition()
094void Body::step(double changet){
095  double aX, aY;
096  // Calc Acceleration
097  aX = netForceX / mass;
098  aY = netForceY / mass;
099  // Calc new velocity
100  xvel = xvel + changet * aX;
101  yvel = yvel + changet * aY;
102  // Calc new position
103  xpos = xpos + changet * xvel;
104  ypos = ypos + changet * yvel;
105  // Set new position
106  setPosition();
107}
108
109// Body-step 3 function that sets the position of the particle
110void Body::setPosition(void){
111  // Place object in correct pixel position
112  double xPixel, yPixel, y;
113  xPixel = xpos / R;
114  xPixel *= wR;
115  yPixel = ypos / R;
```

```
  yPixel *= wR;
  // Add wR to each since origin in SFML is top left
  xPixel += wR;
  yPixel += wR;
  // Make yPixel opposite since SFML plots differently
  if(yPixel > wR){
    y = yPixel - wR;
    yPixel = wR - y;
  }
  else if(yPixel < wR){
    y = wR - yPixel;
    yPixel = wR + y;
  }
  particle.setPosition(sf::Vector2f(xPixel, yPixel));
}
```

1. Assignment: PS4: DNA Sequence Alignment

2. General discussion and what was accomplished:

PS4 called for the computation of the optimal sequence alignment of two DNA strings. Once the theory behind the alignment scores was understood and exercises were completed on paper, dynamic programming was used to come up with the scores for larger files. In other words, subproblems were solved and their solutions stored for later use to, ultimately, solve the original and more complex problem. My program did this successfully as can be seen by Figure 9.

3. One or more key algorithms, data structures, or OO designs central to assignment:

Filling the alignment wasn't too bad. I simply started out by filling the bottom row and rightmost column. This involved place a 0 in the bottom right and then incrementing by 2 to the left and up. Once this was done, I moved from bottom to top and right to left determining the smallest int that could be placed in each box. More specifically, this involved matching the relevant index in the strings to see if they matched and adding 0/1 based on this result to the int diagonal, adding 2 to the lower and right boxes, and then finding which was the smallest. This can be seen below:

```
  // Fill remaining elements
  for(i = sizeX - 1; i > -1; --i){
    for(j = sizeY - 1; j > -1; --j){
      a = matrix[i + 1][j + 1] + penalty(x[i], y[j]);
      b = matrix[i + 1][j] + 2;
      c = matrix[i][j + 1] + 2;
      matrix[i][j] = min(a, b, c);
    }
  }
```

Tracing back how I got to matrix[0][0] was a bit more difficult. My approach entailed working my way back to the bottom right box by starting at matrix[0][0] and, for each iteration, determining how I got to the box I'm moving from. In other words, I checked whether the value in the current box was plus 1 or 0 in relation to the diagonal or plus 2 in relation to the box below or to the right. The if-statement for checking whether the current box value came from the diagonal and is the same can be seen below:

```
    if(i != sizeX && j != sizeY && x[i] == y[j] && matrix[i][j] == matrix[i +
1][j + 1]){
      line += x[i];
      line += ' ';
```

```
      line += y[j];
      line += ' ';
      line += "0\n";
      ++i;
      ++j;
    }
```

4. What was learned:

When it came to this assignment, my understanding of just how useful software can be to other fields greatly increased. Also, I was reminded of Valgrind and how it can be used for analyzing memory.

5. Evidence of success:

```
ok 3 - example10.txt correct
ok 4 - endgaps7.txt correct
ok 5 - fli8.txt correct
ok 6 - fli9.txt correct
ok 7 - fli10.txt correct
ok 8 - ecoli2500.txt correct
ok 9 - ecoli5000.txt correct
ok 10 - ecoli7000.txt correct
ok 11 - ecoli10000.txt correct
```

Figure 9

6. Problems:

Adding to the output string was really annoying me at first, but I figured out that the problem was that I was trying to fit and add too much in one line. It worked fine when I split the elements up.

7. Source code:

Makefile

```
CC = g++
CFLAGS = -Wall -Werror -std=c++0x -g -pedantic
LFLAGS = -lsfml-system
OBJS = main.o ED.o

all: ED

ED: $(OBJS)
    $(CC) $(OBJS) -o ED $(LFLAGS)

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

clean:
    rm *.o ED *~
```

main.cpp

```
#include <iostream>
#include <string>
```

```
#include <SFML/System.hpp>
#include "ED.hpp"

int main(int argc, char* argv[])
{
  sf::Clock clock;
  sf::Time t;
  std::string string1, string2;
  std::cin >> string1;
  std::cin >> string2;

  ED object(string1, string2);
  std::cout << "Edit distance = " << object.OptDistance() << std::endl;
  std::cout << object.Alignment() << std::endl;

  t = clock.getElapsedTime();
  std::cout << "Execution time is " << t.asSeconds() << " seconds \n";

  return 0;
}
```

ED.hpp

```
#pragma once
#ifndef ED_HPP
#define ED_HPP

#include <iostream>
#include <string>
#include <SFML/System.hpp>

class ED
{
public:
  ED(std::string string1, std::string string2);

  ~ED();

  static int penalty(char a, char b);

  static int min(int a, int b, int c);

  int OptDistance();

  std::string Alignment();

private:
  std::string x;
  int sizeX;

  std::string y;
  int sizeY;

  int** matrix;
};

#endif
```

ED.cpp

```
001#include <iostream>
002#include <string>
003#include <SFML/System.hpp>
004#include "ED.hpp"
005
006// Constructor that accepts two strings and allocates memory
007// for the matrix
008ED::ED(std::string string1, std::string string2){
009  x = string1;
010  y = string2;
011  sizeX = x.size();
012  sizeY = y.size();
013  int r = sizeX + 1;
014  int c = sizeY + 1;
015
016  int** m = new int*[r];
017  for(int i = 0; i < r; ++i){
018    m[i] = new int[c];
019  }
020  matrix = m;
021  // Could not use nullptr because using c++0x. Stroustrup uses 0
022  m = 0;
023}
024
025// Destructor that deallocates the memory
026ED::~ED(){
027  int r = sizeX + 1;
028
029  for(int i = 0; i < r; ++i){
030    delete [] matrix[i];
031  }
032
033  delete [] matrix;
034}
035
036// static method that returns penalty for aligning a and b
037int ED::penalty(char a, char b){
038  if(a == b){
039    return 0;
040  }
041  return 1;
042}
043
044// static method that returns min of a, b, and c
045int ED::min(int a, int b, int c){
046  int min = a;
047
048  if(b < min){
049    min = b;
050  }
051  if(c < min){
052    min = c;
053  }
054
055  return min;
056}
057
058// Method that fills the matrix and returns the optimal distance
059int ED::OptDistance(){
```

```
  int i = sizeX;
  int j = sizeY;
  int even = 0;
  // Set bottom row to increasing even numbers right to left
  for(; j > -1; --j, even += 2){
    matrix[i][j] = even;
  }

  // Set right column to increasing even numbers bottom to top
  for(i = sizeX, j = sizeY, even = 0; i > -1; --i, even += 2){
    matrix[i][j] = even;
  }

  int a, b, c;
  // Fill remaining elements
  for(i = sizeX - 1; i > -1; --i){
    for(j = sizeY - 1; j > -1; --j){
      a = matrix[i + 1][j + 1] + penalty(x[i], y[j]);
      b = matrix[i + 1][j] + 2;
      c = matrix[i][j + 1] + 2;
      matrix[i][j] = min(a, b, c);
    }
  }
  // Return optimal distance
  return matrix[0][0];
}

// Method that traces back the matrix and returns a string for
// outputting
std::string ED::Alignment(){
  std::string line;
  int i = 0;
  int j = 0;
  // Due to short-circuit evaluation, do not have to worry about + 1
  while(i < sizeX || j < sizeY){
    if(i != sizeX && j != sizeY && x[i] == y[j] && matrix[i][j] == matrix[i + 1][j + 1]){
      line += x[i];
      line += ' ';
      line += y[j];
      line += ' ';
      line += "0\n";
      ++i;
      ++j;
    }
    else if(i != sizeX && j != sizeY && x[i] != y[j] && matrix[i][j] == matrix[i + 1][j + 1] + 1){
      line += x[i];
      line += ' ';
      line += y[j];
      line += ' ';
      line += "1\n";
      ++i;
      ++j;
    }
    else if(i != sizeX && matrix[i][j] == matrix[i + 1][j] + 2){
      line += x[i];
      line += " - 2\n";
      ++i;
```

```
    }
    else if(j != sizeY && matrix[i][j] == matrix[i][j + 1] + 2){
      line += "- ";
      line += y[j];
      line += " 2\n";
      ++j;
    }
  }

  return line;
}
```

1. Assignment: PS5a: Ring Buffer and Guitar Hero: Ring Buffer implementation with unit tests and exceptions

2. General discussion and what was accomplished:

I basically created an RB class with the necessary member variables and functions. This included a constructor that initializes a vector with the passed in capacity, a size function that returns the number of items, isEmpty and isFull functions that do what their names entail, an enqueue function to add an item to the end, a dequeue function to remove an item from the front, and a peek function that returns, but does not delete, the item at the front. I also ran my code with cpplint.py to make sure it passed the standards and wrote some tests in test.cpp. Exceptions were also included in my Ringbuffer.cpp and test.cpp files.

3. One or more key algorithms, data structures, or OO designs central to assignment:

I think the main key designs were the enqueue and dequeue functions. Enqueue simply involved adding an item to the back of the ring. The body of this function is given below. Basically, the function would throw an exception if the user tried to add an item to a full ring. Otherwise, the new last position was calculated with (last + 1) % cap and the number of items was incremented.

```
if (isFull()) {
  throw std::runtime_error("enqueue: can't enqueue to a full ring.");
}

last = (last + 1) % cap;
vec[last] = x;
numElements++;
```

The primary purpose of the dequeue function was to remove the first element. Before this was done, it was determined whether the ring was empty. If it was, the function would throw an exception. Otherwise, the item was saved, the first position was incremented with the calculation (first + 1) % cap so that it wrapped around if at the end, the number of items was decremented, and then the item was returned.

```
if (isEmpty()) {
  throw std::runtime_error("dequeue: can't dequeue from an empty buffer.");
}

int16_t item;
item = vec[first];
vec[first] = 0;

first = (first + 1) % cap;
```

```
numElements--;
return item;
```

4. What was learned:

The Boost library was again used for this assignment, which increased my comfortability with it further. As it can be seen in Figure 10, my implementation passed all the given tests, as well as my own tests.

5. Evidence of success:

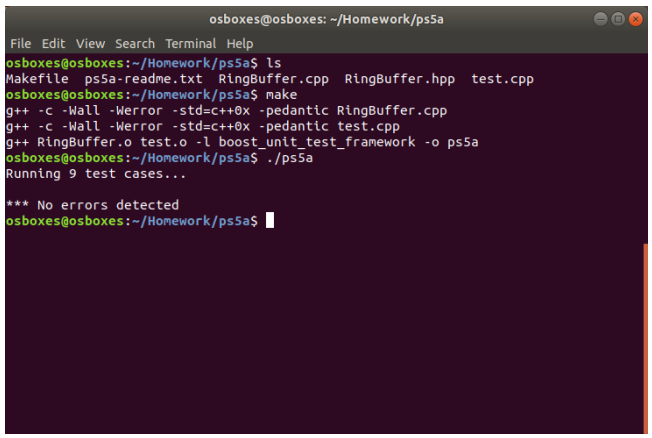

Figure 10

6. Problems:

I did not encounter any serious problems when it came to this program.

7. Source code:

Makefile

```
all: ps5a

ps5a: RingBuffer.o test.o
    g++ RingBuffer.o test.o -l boost_unit_test_framework -o ps5a
RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
    g++ -c -Wall -Werror -std=c++0x -pedantic RingBuffer.cpp
test.o: test.cpp
    g++ -c -Wall -Werror -std=c++0x -pedantic test.cpp
clean:
    rm *.o ps5a *~
```

test.cpp

```
// Copyright 2015 fredm@cs.uml.edu for 91.204 Computing IV
// Wed Mar 25 06:32:17 2015
// Edited by Michael Treacy to include more tests

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

```

```
009#include <stdint.h>
010#include <iostream>
011#include <string>
012#include <exception>
013#include <stdexcept>
014
015#include "RingBuffer.hpp"
016
017BOOST_AUTO_TEST_CASE(RBcontructor) {
018  // normal constructor
019  BOOST_REQUIRE_NO_THROW(RingBuffer(100));
020
021  // this should fail
022  BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
023  BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
024}
025
026BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
027  RingBuffer rb(100);
028
029  rb.enqueue(2);
030  rb.enqueue(1);
031  rb.enqueue(0);
032
033  BOOST_REQUIRE(rb.dequeue() == 2);
034  BOOST_REQUIRE(rb.dequeue() == 1);
035  BOOST_REQUIRE(rb.dequeue() == 0);
036
037  BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
038}
039
040// Tests written by Michael Treacy (tested some of the same things
041// above with slight variations, as well as tested each of the class's
042// other member functions)
043BOOST_AUTO_TEST_CASE(ClassConstructor) {
044  // Generate std::invalid_argument exception on bad constructor
045  BOOST_REQUIRE_THROW(RingBuffer rb1(0), std::invalid_argument);
046
047  // Normal constructor written with class name.
048  // This should not generate exception
049  BOOST_REQUIRE_NO_THROW(RingBuffer rb2(5));
050}
051
052BOOST_AUTO_TEST_CASE(RBenqueue) {
053  RingBuffer rb3(3);
054
055  // Don't generate exception when calling enqueue on buffer
056  // with space
057  BOOST_REQUIRE_NO_THROW(rb3.enqueue(1));
058  BOOST_REQUIRE_NO_THROW(rb3.enqueue(2));
059  BOOST_REQUIRE_NO_THROW(rb3.enqueue(3));
060
061  // Generate std::runtime_error when calling enqueue on
062  // full buffer
063  BOOST_REQUIRE_THROW(rb3.enqueue(4), std::runtime_error);
064}
065
066BOOST_AUTO_TEST_CASE(RBdequeue) {
067  RingBuffer rb4(3);
```

```
068
069  rb4.enqueue(1);
070  rb4.enqueue(2);
071  rb4.enqueue(3);
072
073  // Don't generate exception when calling dequeue on buffer
074  // with items
075  BOOST_REQUIRE_NO_THROW(rb4.dequeue());
076  BOOST_REQUIRE_NO_THROW(rb4.dequeue());
077  BOOST_REQUIRE_NO_THROW(rb4.dequeue());
078
079  // Generate std::runtime_error when calling dequeue on
080  // empty buffer
081  BOOST_REQUIRE_THROW(rb4.dequeue(), std::runtime_error);
082
083  rb4.enqueue(1);
084  rb4.enqueue(2);
085
086  // dequeue returns item from front
087  BOOST_REQUIRE(rb4.dequeue() == 1);
088  BOOST_REQUIRE(rb4.dequeue() == 2);
089}
090
091BOOST_AUTO_TEST_CASE(RBpeek) {
092  RingBuffer rb5(3);
093
094  // Generate std::runtime_error when calling peek on
095  // empty buffer
096  BOOST_REQUIRE_THROW(rb5.peek(), std::runtime_error);
097
098  rb5.enqueue(2);
099  // Buffer size is incremented after item enqueued
100  BOOST_REQUIRE(rb5.size() == 1);
101
102  // Don't generate exception when calling peek on buffer
103  // with item
104  BOOST_REQUIRE_NO_THROW(rb5.peek());
105
106  // peek returns item from front
107  BOOST_REQUIRE(rb5.peek() == 2);
108
109  // peek does not delete item from front
110  BOOST_REQUIRE(rb5.size() == 1);
111}
112
113BOOST_AUTO_TEST_CASE(size) {
114  RingBuffer rb6(3);
115
116  // Buffer size should be 0 when first created
117  BOOST_REQUIRE(rb6.size() == 0);
118
119  rb6.enqueue(1);
120  rb6.enqueue(2);
121  rb6.dequeue();
122
123  // Buffer size should be 1 after two items enqueued
124  // and then one dequeued
125  BOOST_REQUIRE(rb6.size() == 1);
126}
```

```

BOOST_AUTO_TEST_CASE(isEmpty) {
  RingBuffer rb7(3);

  // Buffer should be empty when first created
  BOOST_REQUIRE(rb7.isEmpty() == true);

  rb7.enqueue(1);

  // isEmpty() should return false when buffer has at least
  // one item
  BOOST_REQUIRE(rb7.isEmpty() == false);
}

BOOST_AUTO_TEST_CASE(isFull) {
  RingBuffer rb8(3);

  // Buffer should not be full after first created
  BOOST_REQUIRE(rb8.isFull() == false);

  rb8.enqueue(1);

  // Buffer of capacity 3 should not be full after one item
  // enqueued
  BOOST_REQUIRE(rb8.isFull() == false);

  rb8.enqueue(2);
  rb8.enqueue(3);

  // Buffer of capacity 3 should be full after three items
  // enqueued
  BOOST_REQUIRE(rb8.isFull() == true);
}
```

RingBuffer.hpp

```
// Copyright 2018 Michael Treacy for Computing IV

#pragma once
#ifndef _HOME_OSBOXES_HOMEWORK_PS5A_RINGBUFFER_HPP_
#define _HOME_OSBOXES_HOMEWORK_PS5A_RINGBUFFER_HPP_

#include <stdint.h>
#include <iostream>
#include <vector>
#include <exception>
#include <stdexcept>

class RingBuffer {
 public:
  explicit RingBuffer(int capacity);
  int size();
  bool isEmpty();
  bool isFull();
  void enqueue(int16_t x);
  int16_t dequeue();
  int16_t peek();

 private:
```

```
  std::vector<int16_t> vec;
  int cap;
  int numElements;
  int first;
  int last;
};

#endif  // _HOME_OSBOXES_HOMEWORK_PS5A_RINGBUFFER_HPP_
```

RingBuffer.cpp

```
// Copyright 2018 Michael Treacy for Computing IV

#include "RingBuffer.hpp"
// The header below takes away the cpplint.py complaint
// when .h was used
// #include "/home/osboxes/Homework/ps5a/RingBuffer.h"
#include <stdint.h>
#include <iostream>
#include <vector>
#include <exception>
#include <stdexcept>

// RB constructor that resizes vec based on capacity and
// sets needed member variables
RingBuffer::RingBuffer(int capacity) {
  if (capacity < 1) {
    throw std::invalid_argument("RB constructor: "
    "capacity must be greater than zero.");
  }
  cap = capacity;
  vec.resize(cap);
  numElements = 0;
  first = 0;
  last = -1;
}

// Returns the number of elements
int RingBuffer::size() {
  return numElements;
}

// Determines whether vec is empty or not
bool RingBuffer::isEmpty() {
  if (numElements == 0) {
    return true;
  } else {
    return false;
  }
}

// Determines whether vec is full or not
bool RingBuffer::isFull() {
  if (numElements == cap) {
    return true;
  } else {
    return false;
  }
}
```

```

// Adds item to the end of queue and increases numElements
void RingBuffer::enqueue(int16_t x) {
  if (isFull()) {
    throw std::runtime_error("enqueue: can't enqueue to a full ring.");
  }

  last = (last + 1) % cap;
  vec[last] = x;
  numElements++;
}

// Sets item at beginning of buffer to 0, increments first,
// decrements numElements, and returns removed item
int16_t RingBuffer::dequeue() {
  if (isEmpty()) {
    throw std::runtime_error("dequeue: can't dequeue from an empty buffer.");
  }

  int16_t item;
  item = vec[first];
  vec[first] = 0;

  first = (first + 1) % cap;
  numElements--;
  return item;
}

// Returns element in first position
int16_t RingBuffer::peek() {
  if (isEmpty()) {
    throw std::runtime_error("peek: can't peek from an empty buffer.");
  }
  return vec[first];
}
```

1. Assignment: PS5b: Ring Buffer and Guitar Hero: GuitarHero GuitarString implementation and SFML audio output

2. General discussion and what was accomplished:

This program basically involved implementing the GuitarString class, making sure it passed the GStest.cpp file (Figure 12), and modifying the starter GuitarHeroLite code so that you could use specific keys as if your computer was a piano. I completed the whole assignment and it was done successfully (Figure 13). In other words, every part worked given the fact that the right keys produce sound and working up the key string works up the scale.

3. One or more key algorithms, data structures, or OO designs central to assignment:

One of the key designs that took me a bit of time for this assignment was figuring out how to play the right sound when a certain key was entered. As it can be seen below, I first checked to see if the text was within the correct range of characters. Once this was done, I cast the text and placed it into a char c and then matched it against each key in keys37[] until I found the right sound to play.

```
        if (event.type == sf::Event::TextEntered) {
          if (event.text.unicode < 128) {
```

```
            char c = static_cast<char>(event.text.unicode);
            for (int i = 0; i < 37; i++) {
              if (keys37[i] == c) {
                sounds[i].play();
              }
            }
          }
        }
```

4. What was learned:

The whole assignment increased my understanding of how exceptions can be used in order to test one's code. Along with this, it was very interesting to see how coding can be used in the field of music. It really does seem as if coding touches every type of profession.

5. Evidence of success:

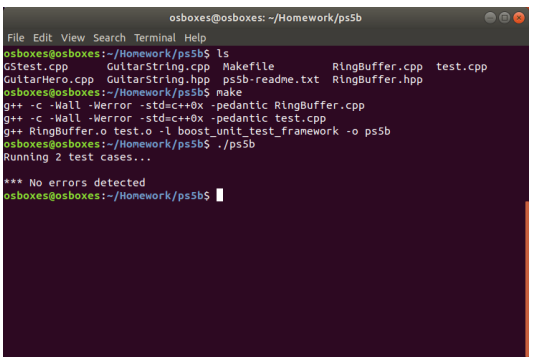

Figure 11

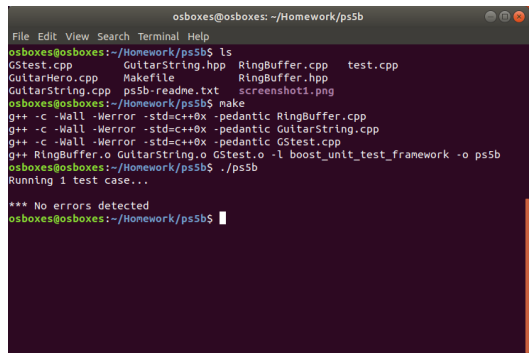

Figure 12

Automatic Score:	12 / 12
Teacher Score:	12 / 12
Days Late:	1
Late Penalty:	0.0%
Current Score:	12.0 / 12

Teacher Notes

Grading Process

Refresh Output

```
== Setting up directory ==

ps5b/
```

Figure 13

6. Problems:

This program took me about 5 hours to complete. The main reason for this was that I initially kept getting a segmentation fault. After staring at my code for 2 hours and using Valgrind, I finally found out that the problem was that my makeSamplesFromString function could not work with just a copy of the GuitarString gs. The object, gs, needed to be passed by reference, or pointer. At first, I just passed by reference with the & symbol, but cpplint.py complained about this so then I decided to pass a pointer. I did not attempt the extra credit parts due to not having the time.

7. Source code:

Makefile

```
all: GuitarHero

GuitarHero: RingBuffer.o GuitarString.o GuitarHero.o
    g++ RingBuffer.o GuitarString.o GuitarHero.o -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio -o GuitarHero
RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
    g++ -c -Wall -Werror -std=c++0x -pedantic RingBuffer.cpp
GuitarString.o: GuitarString.cpp GuitarString.hpp
    g++ -c -Wall -Werror -std=c++0x -pedantic GuitarString.cpp
GuitarHero.o: GuitarHero.cpp
    g++ -c -Wall -Werror -std=c++0x -pedantic GuitarHero.cpp
clean:
    rm *.o GuitarHero *~
```

GuitarHero.cpp

```
// copyright 2018 Michael Treacy for Computing IV
// based on Fred Martin's GuitarHeroLite.cpp

#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>

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

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

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

#define SAMPLES_PER_SEC 44100

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

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

  return samples;
}

int main() {
  sf::RenderWindow window(sf::VideoMode(300, 200), "Guitar Hero");
```

```
39  sf::Event event;
40  std::vector<std::vector<sf::Int16>> samples(37);
41  std::vector<sf::SoundBuffer> buffs(37);
42  std::vector<sf::Sound> sounds(37);
43  std::string keys37 = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/'";
44
45  for (int i = 0; i < 37; i++) {
46    GuitarString gs(440 * pow(2.0, (i - 24.0) / 12.0));
47    samples[i] = makeSamplesFromString(&gs);
48    if (!buffs[i].loadFromSamples(&samples[i][0],
49        samples[i].size(), 2, SAMPLES_PER_SEC))
50      throw std::runtime_error("sf::SoundBuffer: failed to load from samples.");
51    sounds[i].setBuffer(buffs[i]);
52  }
53
54  while (window.isOpen()) {
55    while (window.pollEvent(event)) {
56      if (event.type == sf::Event::Closed) {
57        window.close();
58      } else {
59        if (event.type == sf::Event::TextEntered) {
60          if (event.text.unicode < 128) {
61            char c = static_cast<char>(event.text.unicode);
62            for (int i = 0; i < 37; i++) {
63              if (keys37[i] == c) {
64                sounds[i].play();
65              }
66            }
67          }
68        }
69      }
70
71      window.clear();
72      window.display();
73    }
74  }
75  return 0;
76}
```

GuitarString.hpp

```
01// copyright 2018 Michael Treacy for Computing IV
02
03#pragma once
04#ifndef GUITARSTRING_HPP_
05#define GUITARSTRING_HPP_
06
07#include <SFML/Audio.hpp>
08#include <iostream>
09#include <vector>
10#include "RingBuffer.hpp"
11
12class GuitarString {
13 public:
14  explicit GuitarString(double frequency);
15  explicit GuitarString(std::vector<sf::Int16> init);
16  GuitarString(const GuitarString &obj) {}  // no copy constructor
17  ~GuitarString();
18  void pluck();
```

```
19  void tic();
20  sf::Int16 sample();
21  int time();
22
23 private:
24  RingBuffer* rb;
25  int ticTimes;
26};
27
28#endif  // GUITARSTRING_HPP_
```

GuitarString.cpp

```
01// copyright 2018 Michael Treacy for Computing IV
02
03#include <SFML/Audio.hpp>
04#include <iostream>
05#include <vector>
06#include <random>
07#include <cmath>
08#include "GuitarString.hpp"
09#include "RingBuffer.hpp"
10
11// create a guitar string of the given frequency using a sampling
12// rate of 44,100
13GuitarString::GuitarString(double frequency) {
14  ticTimes = 0;
15  int i = ceil(44100 / frequency);
16  rb = new RingBuffer(i);
17
18  while (rb->isFull() == false) {
19    rb->enqueue(0);
20  }
21}
22
23// create a guitar string with size and initial values are given
24// by the vector
25GuitarString::GuitarString(std::vector<sf::Int16> init) {
26  ticTimes = 0;
27  int i = init.size();
28  rb  = new RingBuffer(i);
29
30  for (int j = 0; rb->isFull() == false; j++) {
31    rb->enqueue(init[j]);
32  }
33}
34
35// destructor
36GuitarString::~GuitarString() {
37  delete rb;
38}
39
40// pluck the guitar string by replacing the buffer with random
41// values, representing white noise
42void GuitarString::pluck() {
43  std::random_device rd;
44  std::mt19937 gen(rd());
45  std::uniform_int_distribution<int> dis(-32768, 32767);
46
```

```
  rb->empty();

  while (rb->isFull() == false) {
    rb->enqueue((sf::Int16)(dis(gen)));
  }
}

// advance the simulation one time step
void GuitarString::tic() {
  ticTimes++;

  sf::Int16 front = rb->dequeue();
  sf::Int16 val = front + rb->peek();
  val = val * 0.5 * 0.996;
  rb->enqueue(val);
}

// return the current sample
sf::Int16 GuitarString::sample() {
  return rb->peek();
}

// return number of times tic was called so far
int GuitarString::time() {
  return ticTimes;
}
```

RingBuffer.hpp

```
// copyright 2018 Michael Treacy for Computing IV

#pragma once
#ifndef RINGBUFFER_HPP_
#define RINGBUFFER_HPP_

#include <stdint.h>
#include <iostream>
#include <exception>
#include <stdexcept>

class RingBuffer {
 public:
  explicit RingBuffer(int capacity);
  ~RingBuffer();
  int size();
  bool isEmpty();
  bool isFull();
  void enqueue(int16_t x);
  int16_t dequeue();
  int16_t peek();
  void empty();

 private:
  int16_t* arry;
  int cap;
  int numElements;
  int first;
  int last;
};
```

```

#endif  // RINGBUFFER_HPP_
```

RingBuffer.cpp

```
// copyright 2018 Michael Treacy for Computing IV

#include "RingBuffer.hpp"
#include <stdint.h>
#include <iostream>
#include <exception>
#include <stdexcept>

// RingBuffer constructor that sets arry based on capacity and
// sets needed member variables
RingBuffer::RingBuffer(int capacity) {
  if (capacity < 1) {
    throw std::invalid_argument("RB constructor: "
    "capacity must be greater than zero.");
  }

  cap = capacity;
  arry = new int16_t[cap];
  numElements = 0;
  first = -1;
  last = -1;
}

// destructor
RingBuffer::~RingBuffer() {
  delete arry;
}

// returns the number of elements
int RingBuffer::size() {
  return numElements;
}

// determines whether arry is empty or not
bool RingBuffer::isEmpty() {
  if (numElements == 0) {
    return true;
  } else {
    return false;
  }
}

// determines whether arry is full or not
bool RingBuffer::isFull() {
  if (numElements == cap) {
    return true;
  } else {
    return false;
  }
}

// adds item to the end of queue and increases numElements
void RingBuffer::enqueue(int16_t x) {
  if (isFull()) {
```

```
    throw std::runtime_error("enqueue: can't enqueue to a full ring.");
  }

  if (first == -1) {
    first = 0;
  }

  last = (last + 1) % cap;
  arry[last] = x;
  numElements++;
}

// removes item at front of arry
int16_t RingBuffer::dequeue() {
  if (isEmpty()) {
    throw std::runtime_error("dequeue: can't dequeue from an empty buffer.");
  }

  int16_t item = arry[first];

  if (first == last) {
    first = -1;
    last = -1;
  } else {
    first = (first + 1) % cap;
  }

  numElements--;
  return item;
}

// returns element in first position
int16_t RingBuffer::peek() {
  if (isEmpty()) {
    throw std::runtime_error("peek: can't peek from an empty buffer.");
  }

  return arry[first];
}

void RingBuffer::empty() {
  numElements = 0;
  first = -1;
  last = -1;
}
```

GStest.cpp

```
/*
  Copyright 2015 Fred Martin, fredm@cs.uml.edu
  Wed Apr  1 09:43:12 2015
  test file for GuitarString class

  compile with
  g++ -c GStest.cpp -lboost_unit_test_framework
  g++ GStest.o GuitarString.o RingBuffer.o -o GStest -lboost_unit_test_framework
*/

#define BOOST_TEST_DYN_LINK
```

```
12#define BOOST_TEST_MODULE Main
13#include <boost/test/unit_test.hpp>
14
15#include <vector>
16#include <exception>
17#include <stdexcept>
18
19#include "GuitarString.hpp"
20
21BOOST_AUTO_TEST_CASE(GS) {
22  std::vector<sf::Int16> v;
23
24  v.push_back(0);
25  v.push_back(2000);
26  v.push_back(4000);
27  v.push_back(-10000);
28
29  BOOST_REQUIRE_NO_THROW(GuitarString gs = GuitarString(v));
30
31  GuitarString gs = GuitarString(v);
32
33  // GS is 0 2000 4000 -10000
34  BOOST_REQUIRE(gs.sample() == 0);
35
36  gs.tic();
37  // it's now 2000 4000 -10000 996
38  BOOST_REQUIRE(gs.sample() == 2000);
39
40  gs.tic();
41  // it's now 4000 -10000 996 2988
42  BOOST_REQUIRE(gs.sample() == 4000);
43
44  gs.tic();
45  // it's now -10000 996 2988 -2988
46  BOOST_REQUIRE(gs.sample() == -10000);
47
48  gs.tic();
49  // it's now 996 2988 -2988 -4483
50  BOOST_REQUIRE(gs.sample() == 996);
51
52  gs.tic();
53  // it's now 2988 -2988 -4483 1984
54  BOOST_REQUIRE(gs.sample() == 2988);
55
56  gs.tic();
57  // it's now -2988 -4483 1984 0
58  BOOST_REQUIRE(gs.sample() == -2988);
59
60  // a few more times
61  gs.tic();
62  BOOST_REQUIRE(gs.sample() == -4483);
63  gs.tic();
64  BOOST_REQUIRE(gs.sample() == 1984);
65  gs.tic();
66  BOOST_REQUIRE(gs.sample() == 0);
67}
```

test.cpp

```
01// Copyright 2015 fredm@cs.uml.edu for 91.204 Computing IV
02// Wed Mar 25 06:32:17 2015
03
04#define BOOST_TEST_DYN_LINK
05#define BOOST_TEST_MODULE Main
06#include <boost/test/unit_test.hpp>
07
08#include <stdint.h>
09#include <iostream>
10#include <string>
11#include <exception>
12#include <stdexcept>
13
14#include "RingBuffer.hpp"
15
16BOOST_AUTO_TEST_CASE(RBcontructor) {
17  // normal constructor
18  BOOST_REQUIRE_NO_THROW(RingBuffer(100));
19
20  // this should fail
21  BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
22  BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
23}
24
25BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
26  RingBuffer rb(100);
27
28  rb.enqueue(2);
29  rb.enqueue(1);
30  rb.enqueue(0);
31
32  BOOST_REQUIRE(rb.dequeue() == 2);
33  BOOST_REQUIRE(rb.dequeue() == 1);
34  BOOST_REQUIRE(rb.dequeue() == 0);
35
36  BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
37}
```

1. Assignment: PS6: Airport Simulation Project (C++11 Concurrency)

2. General discussion and what was accomplished:

This assignment basically simulates the landings of 7 airplanes at Logan Airport. This was done by using mutexes, a condition_variable, and boolean variables for each runway. With the help of these different variables, I was able to add the necessary mutual exclusion and synchronization needed to avoid any airplane crashes.

3. One or more key algorithms, data structures, or OO designs central to assignment:

The two main algorithms that were central to the assignment were located under the synchronization comments in AirportServer.cpp's functions, reserveRunway and releaseRunway. More specifically, I implemented certain rules in reserveRunway, such as runway 9 never being used simultaneously with 4R or 15R, by checking which runway name I had, locking its associated mutex with a unique_lock, putting the thread on hold until the needed boolean runways were open, setting the boolean runways to closed, and then unlocking. My synchronization for the releaseRunway function simply involved

checking which runway I had, setting its associated closed runways to open, or true, and then using notify_one. I should also mention that I used a variable called requests to keep the number of requests below 7.

The code for runway "4L" in the reserveRunway function can be seen below. As stated above, my reserveRunway function checked to see which runway was passed in, locked its associated mutex, waited until all the needed runways were open, set them all to closed, and unlocked.

```
            requests++;
            string str = AirportRunways::runwayName(runway);
            if (str == "4L") {
              unique_lock<mutex> l1(m4L);
              cV.wait(l1, [this] () { return (b4L == true && b15L == true && b15R
== true); });
              b4L = false;
              b15L = false;
              b15R = false;
              l1.unlock();
            }
```

The code for runway "4L" in the releaseRunway function follows. As stated before, my releaseRunway function checked which runway was passed in and set its associated runways that were closed to open, or true.

```
          string str = AirportRunways::runwayName(runway);
          if (str == "4L") {
            b4L = true;
            b15L = true;
            b15R = true;
          }
```

Later in the code, notify_one was called so that a thread waiting on the runway(s) could continue. I should also mention the lamda I wrote in Airport.cpp. This was a quick fix to the original starter code, but it was a requirement of the assignment. As it can be seen below, ap was used to called land().

```
          apths.push_back(thread([ap] () { ap->land(); }));
```

4. What was learned:

I definitely feel a bit more comfortable with threads now. The concepts seemed straightforward at first, but immediately after my initial attempts at this assignment I realized that I did not grasp concurrency as well as I thought. It was only through repeated failures and reading documentations for threads, mutexes, and condition_variables that I finally was able to get my program working.

5. Evidence of success:

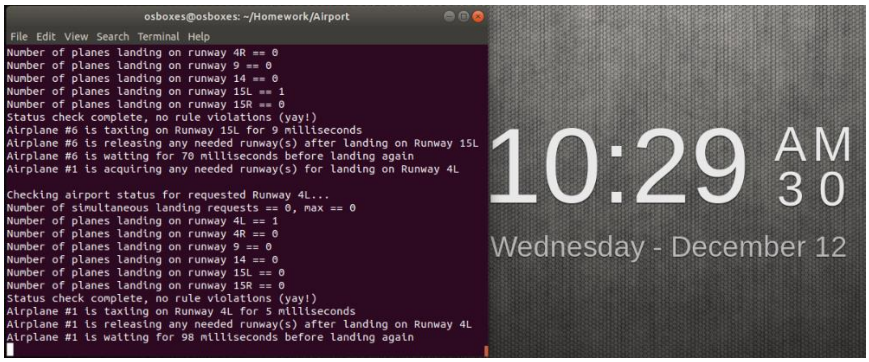

Figure 14

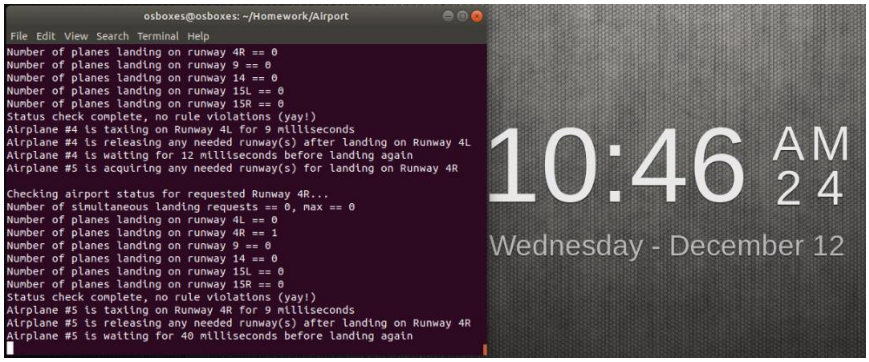

Figure 15 (Still running after over 15 minutes)

Assignment	Airport
Student	Treacy Michael
Submission Time	2018-12-02 03:31:20 UTC
Submitted File	MichaelTreacyAirport.tar.gz
Student Notes	My executable is named Airport and not Airport-Nosync.

Score

Automatic Score:	no data / 22
Teacher Score:	22 / 22

Figure 16

6. Problems:

This assignment took me more time than I thought it would. I had to read through several tutorials to better comprehend mutexes and condition_variables, especially how to correctly use the wait function. After about 3 hours of reading, I found that the code wasn't too difficult to write.

7. Source code:

Makefile

```
CC = g++
CFLAGS = -c -g -Og -std=c++11
OBJ = Airplane.o Airport.o AirportRunways.o AirportServer.o
DEPS =
LIBS = -pthread
EXE = Airport
```

```
07
08 all: $(OBJ)
09     $(CC) $(OBJ) -o $(EXE) $(LIBS)
10
11 %.o: %.cpp $(DEPS)
12     $(CC) $(CFLAGS) -o $@ $<
13
14 clean:
15     rm -f $(OBJ) $(EXE) *~
```

Airport.cpp

```
01/**
02*  Airport driver program
03*/
04
05#include <iostream>
06#include <thread>
07#include <vector>
08
09#include "AirportServer.h"
10#include "AirportRunways.h"
11#include "Airplane.h"
12
13using namespace std;
14
15
16// void run(Airplane* ap)
17// {
18//  ap->land();
19
20// } // end run
21
22
23int main(void)
24{
25    AirportServer as;
26
27    vector<thread> apths; // Airplane threads
28
29                                    // Create and launch the individual Airplane
threads
30    for (int i = 1; i <= AirportRunways::NUM_AIRPLANES; i++)
31    {
32          Airplane* ap = new Airplane(i, &as);
33
34          //apths.push_back(thread(&run, ap));
35          apths.push_back(thread([ap] () { ap->land(); }));
36    }
37
38    // Wait for all Airplane threads to terminate (shouldn't happen!)
39    for (auto& th : apths)
40    {
41          th.join();
42    }
43
44    return 0;
45
46} // end main
```

Airplane.h

```
/**
*  Airplane.h
*  Definition of the Airplane class
*/

#ifndef AIRPLANE_H
#define AIRPLANE_H

#include "AirportRunways.h"
#include "AirportServer.h"


class Airplane
{
public:

    int airplaneNum;
    AirportServer* apServ;

    // Value constructor for the Airplane class
    Airplane(int num, AirportServer* s)
    {
          airplaneNum = num;
          apServ = s;
    }


    // Setter method for requestedRunway
    void setRequestedRunway(AirportRunways::RunwayNumber runway)
    {
          requestedRunway = runway;
    }


    // The run() function for Airplane threads in Airport will call this function
    void land();


private:

    AirportRunways::RunwayNumber requestedRunway; // Picked at random

}; // end class Airplane

#endif
```

Airplane.cpp

```
#include <random>
#include <thread>
#include <chrono>

#include "Airplane.h"

// The run() function in Airport will call this function
void Airplane::land()
{
```

```
    // obtain a seed from the system clock:
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();

    std::default_random_engine generator(seed);
    std::uniform_int_distribution<int> runwayNumberDistribution(AirportRun-
ways::RUNWAY_4L, AirportRunways::RUNWAY_15R);

    while (true)
    {
          // Get ready to land
          requestedRunway = AirportRunways::RunwayNumber(runwayNumberDistribu-
tion(generator));

          apServ->reserveRunway(airplaneNum, requestedRunway);

          // Landing complete
          apServ->releaseRunway(airplaneNum, requestedRunway);

          // Wait on the ground for a while (to prevent starvation of other air-
planes)
          std::this_thread::sleep_for(std::chrono::milliseconds(1000));

    } // end while

} // end Airplane::land
```

AirportRunways.h

```
/**
*  Class AirportRunways provides definitions of constants and helper methods for
the Airport simulation.
*/

#ifndef AIRPORT_RUNWAYS_H
#define AIRPORT_RUNWAYS_H

#include <iostream>
#include <string>
#include <mutex>

using namespace std;


class AirportRunways
{
public:

    static const int NUM_RUNWAYS = 6;     // Number of runways in this simulation
    static const int NUM_AIRPLANES = 7;   // Number of airplanes in this simula-
tion
    static const int MAX_LANDING_REQUESTS = 6; // Maximum number of simultaneous
landing requests that Air Traffic Control can handle

    enum RunwayNumber { RUNWAY_4L, RUNWAY_4R, RUNWAY_9, RUNWAY_14, RUNWAY_15L,
RUNWAY_15R };

    static mutex checkMutex; // enforce mutual exclusion on checkAirportStatus

    static string runwayName(RunwayNumber rn);
```

```
28
29    /**
30    *  Check the status of the aiport with respect to any violation of the rules.
31    */
32    static void checkAirportStatus(RunwayNumber requestedRunway);
33
34    /**
35    *  requestRunway() and finishedWithRunway() are helper methods for keeping
track of the airport status
36    */
37
38    static void requestRunway(RunwayNumber rn)
39    {
40          runwayInUse[rn]++;
41
42    } // end useRunway()
43
44
45    static void finishedWithRunway(RunwayNumber rn)
46    {
47          runwayInUse[rn]--;
48
49    } // end finishedWithRunway()
50
51
52    static int getNumLandingRequests()
53    {
54          return numLandingRequests;
55    }
56
57
58    static void incNumLandingRequests()
59    {
60          numLandingRequests++;
61          if (numLandingRequests > maxNumLandingRequests)
62                maxNumLandingRequests = numLandingRequests;
63    }
64
65
66    static void decNumLandingRequests()
67    {
68          numLandingRequests--;
69    }
70
71private:
72
73    /**
74     *  The following variables and methods are used to detect violations of the
rules of this simulation.
75     */
76
77    static int runwayInUse[NUM_RUNWAYS]; // Keeps track of how many airplanes are
attempting to land on a given runway
78
79    static int numLandingRequests; // Keeps track of the number of simultaneous
landing requests
80
81    static int maxNumLandingRequests; // Keeps track of the max number of simul-
taneous landing requests
```

```
82
83}; // end class AirportRunways
84
85#endif
```

AirportRunways.cpp

```
01#include "AirportRunways.h"
02
03int AirportRunways::runwayInUse[AirportRunways::NUM_RUNWAYS];
04
05int AirportRunways::numLandingRequests = 0;
06
07int AirportRunways::maxNumLandingRequests = 0;
08
09mutex AirportRunways::checkMutex;
10
11
12string AirportRunways::runwayName(RunwayNumber rn)
13{
14    switch (rn)
15    {
16    case RUNWAY_4L:
17          return "4L";
18    case RUNWAY_4R:
19          return "4R";
20    case RUNWAY_9:
21          return "9";
22    case RUNWAY_14:
23          return "14";
24    case RUNWAY_15L:
25          return "15L";
26    case RUNWAY_15R:
27          return "15R";
28    default:
29          return "Unknown runway " + rn;
30    } // end switch
31
32} // end AirportRunways::runwayName()
33
34
35 /**
36  *  Check the status of the aiport with respect to any violation of the rules.
37  */
38void AirportRunways::checkAirportStatus(RunwayNumber requestedRunway)
39{
40    lock_guard<mutex> checkLock(checkMutex);
41
42    bool crash = false; // Set to true if any rule is violated
43
44    cout << "\nChecking airport status for requested Runway " << runwayName(re-
questedRunway) << "..." << endl;
45
46    requestRunway(requestedRunway);
47
48    // Check the number of landing requests
49    cout << "Number of simultaneous landing requests == " << numLandingRequests
50          << ", max == " << maxNumLandingRequests << endl;
51
```

```
    if (numLandingRequests > MAX_LANDING_REQUESTS)
    {
          cout << "***** The number of simultaneous landing requests exceeds Air Traffic Control limit of " << MAX_LANDING_REQUESTS << "!\n";
          crash = true;
    }

    // Check the occupancy of each runway
    for (int i = RUNWAY_4L; i <= RUNWAY_15R; i++)
    {
          cout << "Number of planes landing on runway " << runwayName(RunwayNumber(i)) << " == " << runwayInUse[i] << endl;

          if (runwayInUse[i] > 1)
          {
                cout << "***** The number of planes landing on runway " << runwayName(RunwayNumber(i)) << " is greater than 1!\n";
                crash = true;
          }
    }

    // Check individual restrictions on each runway
    if ((runwayInUse[RUNWAY_9] > 0)
          && ((runwayInUse[RUNWAY_4R] > 0) || (runwayInUse[RUNWAY_15R] > 0)))
    {
          cout << "***** Runways 9, 4R, and/or 15R may not be used simultaneously!\n";
          crash = true;
    }

    if (((runwayInUse[RUNWAY_15L] > 0) || (runwayInUse[RUNWAY_15R] > 0))
          && ((runwayInUse[RUNWAY_4L] > 0) || (runwayInUse[RUNWAY_4R] > 0)))
    {
          cout << "***** Runways 15L or 15R may not be used simultaneously with Runways 4L or 4R!\n";
          crash = true;
    }

    // If any of the rules have been violated, terminate the simulation
    if (crash)
    {
          cout << "***** CRASH! One or more rules have been violated. Due to the crash, the airport is closed!\n";
          exit(-1); // Abnormal program termination
    }

    // Status check is normal
    cout << "Status check complete, no rule violations (yay!)\n";

} // end AirportRunways::checkAirportStatus()
```

AirportServer.h

```
/**
*  AirportServer.h
*  This class defines the methods called by the Airplanes
*/

#ifndef AIRPORT_SERVER_H
```

```
07#define AIRPORT_SERVER_H
08
09#include <mutex>
10#include <random>
11#include <condition_variable>
12
13#include "AirportRunways.h"
14
15
16
17class AirportServer
18{
19public:
20
21    /**
22    *  Default constructor for AirportServer class
23    */
24    AirportServer()
25    {
26          // ***** Initialize any Locks and/or Condition Variables here as neces-
sary *****
27      requests = 0;
28      b4L = true;
29      b4R = true;
30      b9 = true;
31      b14 = true;
32      b15L = true;
33      b15R = true;
34    } // end AirportServer default constructor
35
36
37      /**
38      *  Called by an Airplane when it wishes to land on a runway
39      */
40    void reserveRunway(int airplaneNum, AirportRunways::RunwayNumber runway);
41
42    /**
43    *  Called by an Airplane when it is finished landing
44    */
45    void releaseRunway(int airplaneNum, AirportRunways::RunwayNumber runway);
46
47
48private:
49
50    // Constants and Random number generator for use in Thread sleep calls
51    static const int MAX_TAXI_TIME = 10; // Maximum time the airplane will occupy
the requested runway after landing, in milliseconds
52    static const int MAX_WAIT_TIME = 100; // Maximum time between landings, in
milliseconds
53
54    /**
55    *  Declarations of mutexes and condition variables
56    */
57    mutex runwaysMutex; // Used to enforce mutual exclusion for acquiring & re-
leasing runways
58
59    /**
60     *  ***** Add declarations of your own Locks and Condition Variables here
*****
```

```
61     */
62    int requests;
63    bool b4L;
64    bool b4R;
65    bool b9;
66    bool b14;
67    bool b15L;
68    bool b15R;
69
70    mutex m4L;
71    mutex m4R;
72    mutex m9;
73    mutex m14;
74    mutex m15L;
75    mutex m15R;
76
77    condition_variable cV;
78}; // end class AirportServer
79
80#endif
```

AirportServer.cpp

```
001#include <iostream>
002#include <thread>
003#include <condition_variable>
004
005#include "AirportServer.h"
006
007
008/**
009*  Called by an Airplane when it wishes to land on a runway
010*/
011void AirportServer::reserveRunway(int airplaneNum, AirportRunways::RunwayNumber
runway)
012{
013   // Acquire runway(s)
014   {  // Begin critical region
015
016     // unique_lock<mutex> runwaysLock(runwaysMutex);
017
018         {
019               lock_guard<mutex> lk(AirportRunways::checkMutex);
020
021               cout << "Airplane #" << airplaneNum << " is acquiring any needed
runway(s) for landing on Runway "
022                       << AirportRunways::runwayName(runway) << endl;
023         }
024
025         // ***** Add your synchronization here! *****
026         if (requests < 7) {
027           requests++;
028           string str = AirportRunways::runwayName(runway);
029           if (str == "4L") {
030             unique_lock<mutex> l1(m4L);
031             cV.wait(l1, [this] () { return (b4L == true && b15L == true && b15R
== true); });
032             b4L = false;
033             b15L = false;
```

```
            b15R = false;
            l1.unlock();
          } else if (str == "4R") {
            unique_lock<mutex> l2(m4R);
            cV.wait(l2, [this] () { return (b4R == true && b15L == true && b15R == true && b9 == true); });
            b4R = false;
            b15L = false;
            b15R = false;
            b9 = false;
            l2.unlock();
          } else if (str == "9") {
            unique_lock<mutex> l3(m9);
            cV.wait(l3, [this] () { return (b9 == true && b15R == true && b4R == true); });
            b9 = false;
            b15R = false;
            b4R = false;
            l3.unlock();
          } else if (str == "14") {
            unique_lock<mutex> l4(m14);
            cV.wait(l4, [this] () { return (b14 == true); });
            b14 = false;
            l4.unlock();
          } else if (str == "15L") {
            unique_lock<mutex> l5(m15L);
            cV.wait(l5, [this] () { return (b15L == true && b4L == true && b4R == true); });
            b15L = false;
            b4L = false;
            b4R = false;
            l5.unlock();
          } else if (str == "15R") {
            unique_lock<mutex> l6(m15R);
            cV.wait(l6, [this] () { return (b15R == true && b4L == true && b4R == true && b9 == true); });
            b15R = false;
            b4L = false;
            b4R = false;
            b9 = false;
            l6.unlock();
          }
        }

        // Check status of the airport for any rule violations
        AirportRunways::checkAirportStatus(runway);

        // runwaysLock.unlock();

    } // End critical region

    // obtain a seed from the system clock:
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::default_random_engine generator(seed);

    // Taxi for a random number of milliseconds
    std::uniform_int_distribution<int> taxiTimeDistribution(1, MAX_TAXI_TIME);
    int taxiTime = taxiTimeDistribution(generator);

```

```
089   {
090         lock_guard<mutex> lk(AirportRunways::checkMutex);
091
092         cout << "Airplane #" << airplaneNum << " is taxiing on Runway " << Air-
portRunways::runwayName(runway)
093                << " for " << taxiTime << " milliseconds\n";
094   }
095
096   std::this_thread::sleep_for(std::chrono::milliseconds(taxiTime));
097} // end AirportServer::reserveRunway()
098
099
100 /**
101  *  Called by an Airplane when it is finished landing
102  */
103void AirportServer::releaseRunway(int airplaneNum, AirportRunways::RunwayNumber
runway)
104{
105   // Release the landing runway and any other needed runways
106   { // Begin critical region
107
108     // unique_lock<mutex> runwaysLock(runwaysMutex);
109
110         {
111               lock_guard<mutex> lk(AirportRunways::checkMutex);
112
113               cout << "Airplane #" << airplaneNum << " is releasing any needed
runway(s) after landing on Runway "
114                     << AirportRunways::runwayName(runway) << endl;
115         }
116
117         // ***** Add your synchronization here! *****
118         string str = AirportRunways::runwayName(runway);
119         if (str == "4L") {
120           b4L = true;
121           b15L = true;
122           b15R = true;
123         } else if (str == "4R") {
124           b4R = true;
125           b15L = true;
126           b15R = true;
127           b9 = true;
128         } else if (str == "9") {
129           b9 = true;
130           b15R = true;
131           b4R = true;
132         } else if (str == "14") {
133           b14 = true;
134         } else if (str == "15L") {
135           b15L = true;
136           b4L = true;
137           b4R = true;
138         } else if (str == "15R") {
139           b15R = true;
140           b4L = true;
141           b4R = true;
142           b9 = true;
143         }
144
```

```
        // Update the status of the airport to indicate that the landing is complete
        AirportRunways::finishedWithRunway(runway);

        // runwaysLock.unlock();
        requests--;
        cV.notify_one();

  } // End critical region

  // obtain a seed from the system clock:
  unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
  std::default_random_engine generator(seed);

  // Wait for a random number of milliseconds before requesting the next landing for this Airplane
  std::uniform_int_distribution<int> waitTimeDistribution(1, MAX_WAIT_TIME);
  int waitTime = waitTimeDistribution(generator);

  {
        lock_guard<mutex> lk(AirportRunways::checkMutex);

        cout << "Airplane #" << airplaneNum << " is waiting for " << waitTime << " milliseconds before landing again\n";
  }

  std::this_thread::sleep_for(std::chrono::milliseconds(waitTime));

} // end AirportServer::releaseRunway()
```

1. Assignment: PS7a Kronos Time Clock: Introduction to Regular Expressions Parsing

2. General discussion and what was accomplished:

This project involved using regular expressions to find boots and boot completions in Kronos log files. I did this by, first, using an online regex tester to try and create the required regex's for the boot and boot completion. I then created a for-loop that went through the log file and wrote the boots, incorrect boots, and boot completions to an output file. This involved matching my regex's to each line in the source log file and checking whether or not a boot or boot completion was found. If one was found, the necessary code would execute. In other words, if there was a boot, the line number would be reported, the date and time, as well as the text, "Boot Start". If there was a double boot, "**** Incomplete boot ****" was written to the output file. Lastly, if a boot completion was found, the line number, the data and time, "Boot Complete", and the boot time was written. The end of the program simply involved closing the files. As it can be seen in Figure 17, my output files were correct.

3. One or more key algorithms, data structures, or OO designs central to assignment:

The immediate code below shows how I computed the total milliseconds between a boot and boot completion:

```
      date d1(from_simple_string(bStartDate));
      date d2(from_simple_string(bEndDate));
      tDuration start(bStartTime);
      ptime t1(d1, time_duration(start.getHours(), start.getMinutes(),
                                 start.getSeconds(), 0));
```

```
088      tDuration end(bEndTime);
089      ptime t2(d2, time_duration(end.getHours(), end.getMinutes(),
090                                 end.getSeconds(), 0));
091      time_duration td = t2 - t1;
092      // Lastly, output data
093      oFile << i << '(' << inputFileName << "): ";
094      oFile << bEndDate << ' ' << bEndTime << " Boot Completed\n";
095      oFile << "        Boot Time: " << td.total_milliseconds() << "ms\n\n";
096      booted = false;
```

As it can be seen, I created a date from each date string and created two ptime's. It is important to state that I created a class called tDuration for this since I needed to get the number of hours, minutes, and seconds from a string. The code for this can be seen in what follows:

```
119tDuration::tDuration(string line) {
120  char c;
121  hours = 0;
122  minutes = 0;
123  seconds = 0;
124
125  c = line[1];
126  hours  = c - '0';
127  c = line[0];
128  hours += (c - '0') * 10;
129
130  c = line[4];
131  minutes = c - '0';
132  c = line[3];
133  minutes += (c - '0') * 10;
134
135  c = line[7];
136  seconds = c - '0';
137  c = line[6];
138  seconds += (c - '0') * 10;
139}
```

Basically, I knew the hours would be in the first two positions, the minutes in position 3 and 4, and the seconds in 6 and 7. After the ptime's were made and t1 subtracted from t2, I was able to get the total milliseconds and write to the output file.

The two regex's I created are included below:

```
044  regex e1(".*log.c.166. server started.*");
045  regex e2(".*oejs.AbstractConnector:Started"
046           " SelectChannelConnector@0.0.0.0:9080.*");
```

The first regex, e1, basically allows one to later search for log.c.166. server started. The .*'s at the beginning and end make it so 0 or more characters can appear before and after the required characters. I did this because all that was needed when it came to searching was the key text and shortening the regex made the code nicer to read. Finding the date and time for computing afterwards was trivial. In the files, the . after the second 6 is actually a), but I used a . because a . accepts any character besides a newline. For regex e2, I carried on with the same logic of accepting 0 or more characters before and after the required characters. This time oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080 was the text I required. After these regex's were initialized, I could use regex_match to match a read-in line from the log to each regex and proceed with computations and writing to the output file if necessary.

4. What was learned:

I have actually never used regular expressions before this class. I know a number of students have, but this was all new to me. With that said, it can be said that I learned of a great tool to parse files and search for certain keywords. Luckily, it seems as though the syntax for regular expressions is the same for numerous languages.

5. Evidence of success:

```
ok 15 - device1_intouch.log.rpt matches sample
ok 16 - device2_intouch.log.rpt matches sample
ok 17 - device3_intouch.log.rpt matches sample
ok 18 - device4_intouch.log.rpt matches sample
ok 19 - device5_intouch.log.rpt matches sample
ok 20 - device6_intouch.log.rpt matches sample
```

Figure 17

6. Problems:

I did not have too many problems with this last assignment. Bottlenose complained a bit about my use of the stoi function, but I decided to convert my char's to int's simply by subtracting '0' from them and then multiplying by 10 or 100 if needs be.

7. Source code:

Makefile

```
all: ps7a

ps7a: main.o
    g++ main.o -lboost_regex -lboost_date_time -o ps7a
main.o: main.cpp
    g++ -c -Wall -Werror -std=c++0x -pedantic main.cpp
clean:
    rm *.o ps7a *~
```

main.cpp

```
// Copyright 2018 Michael Treacy, MICHAEL_TREACY@student.uml.edu
// Fri Dec 7, 2018

#include <boost/regex.hpp>
#include <iostream>
#include <string>
#include <fstream>
#include "boost/date_time/gregorian/gregorian.hpp"
#include "boost/date_time/posix_time/posix_time.hpp"

using std::string;
using std::ifstream;
using std::ofstream;

using boost::regex;
```

```
016using boost::regex_match;
017using boost::gregorian::date;
018using boost::gregorian::from_simple_string;
019using boost::gregorian::date_period;
020using boost::gregorian::date_duration;
021
022using boost::posix_time::ptime;
023using boost::posix_time::time_duration;
024
025class tDuration {
026 public:
027  tDuration();
028  explicit tDuration(string line);
029  int getHours() { return hours; }
030  int getMinutes() { return minutes; }
031  int getSeconds() { return seconds; }
032 private:
033  int hours;
034  int minutes;
035  int seconds;
036};
037
038int main(int argc, char* argv[]) {
039  string inputFileName = argv[1];
040  ifstream iFile(inputFileName);
041  string outputFileName = inputFileName + ".rpt";
042  ofstream oFile(outputFileName);
043
044  regex e1(".*log.c.166. server started.*");
045  regex e2(".*oejs.AbstractConnector:Started"
046           " SelectChannelConnector@0.0.0.0:9080.*");
047  string line;
048  bool booted = false;
049  string bStartDate;
050  string bStartTime;
051
052  // for-loop that reads through the entire file
053  for (int i = 1; getline(iFile, line); i++) {
054    if (regex_match(line, e1)) {
055      if (booted) {
056        oFile << "**** Incomplete boot ****\n\n";
057      }
058      // If matches e1, collect start date and time
059      oFile << "=== Device boot ===\n";
060      bStartDate.clear();
061      for (int w = 0; w < 10; w++) {
062        bStartDate += line[w];
063      }
064      bStartTime.clear();
065      for (int x = 11; x < 19; x++) {
066        bStartTime += line[x];
067      }
068      // Also, output data
069      oFile << i << '(' << inputFileName << "): ";
070      oFile << bStartDate << ' ' << bStartTime<< " Boot Start\n";
071      booted = true;
072    } else if (regex_match(line, e2)) {
073      string bEndDate;
074      for (int y = 0; y < 10; y++) {
```

```
075         bEndDate += line[y];
076       }
077       string bEndTime;
078       for (int z = 11; z < 19; z++) {
079         bEndTime += line[z];
080       }
081       // If matches e2, collect end date and time (done above)
082       // and compute the total milliseconds of boot sequence
083       date d1(from_simple_string(bStartDate));
084       date d2(from_simple_string(bEndDate));
085       tDuration start(bStartTime);
086       ptime t1(d1, time_duration(start.getHours(), start.getMinutes(),
087                                  start.getSeconds(), 0));
088       tDuration end(bEndTime);
089       ptime t2(d2, time_duration(end.getHours(), end.getMinutes(),
090                                  end.getSeconds(), 0));
091       time_duration td = t2 - t1;
092       // Lastly, output data
093       oFile << i << '(' << inputFileName << "): ";
094       oFile << bEndDate << ' ' << bEndTime << " Boot Completed\n";
095       oFile << "        Boot Time: " << td.total_milliseconds() << "ms\n\n";
096       booted = false;
097     }
098   }
099
100   if (booted == true) {
101     oFile << "**** Incomplete boot ****\n\n";
102   }
103
104   iFile.close();
105   oFile.close();
106
107   return 0;
108 }
109
110 // Default constructor that sets each member to 0
111 tDuration::tDuration() {
112   hours = 0;
113   minutes = 0;
114   seconds = 0;
115 }
116
117 // Constructor that takes a date as a string and determines
118 // its private variables based on that string
119 tDuration::tDuration(string line) {
120   char c;
121   hours = 0;
122   minutes = 0;
123   seconds = 0;
124
125   c = line[1];
126   hours  = c - '0';
127   c = line[0];
128   hours += (c - '0') * 10;
129
130   c = line[4];
131   minutes = c - '0';
132   c = line[3];
133   minutes += (c - '0') * 10;
```

```
134
135  c = line[7];
136  seconds = c - '0';
137  c = line[6];
138  seconds += (c - '0') * 10;
139}
```