Lucas Faletra
COMP IV: Project Portfolio
Spring 2019

Contents:

PS0: Hello World With SFML

Assignment:

The objective of this assignment was to become familiar with the C++ library SFML by creating a simple project. The first part was to run the SFML demo code for displaying a circle to check if SFML was configured correctly. After this was working, the next task was to create a moving sprite constructed from an image that responds to keystrokes. I accomplished this by taking an image of an optical illusion, constructing the sprite from the image, and incrementing the coordinates in a loop so that the sprite appeared to be moving. Pressing a key will increment the framerate, which makes the image appear to be moving faster.

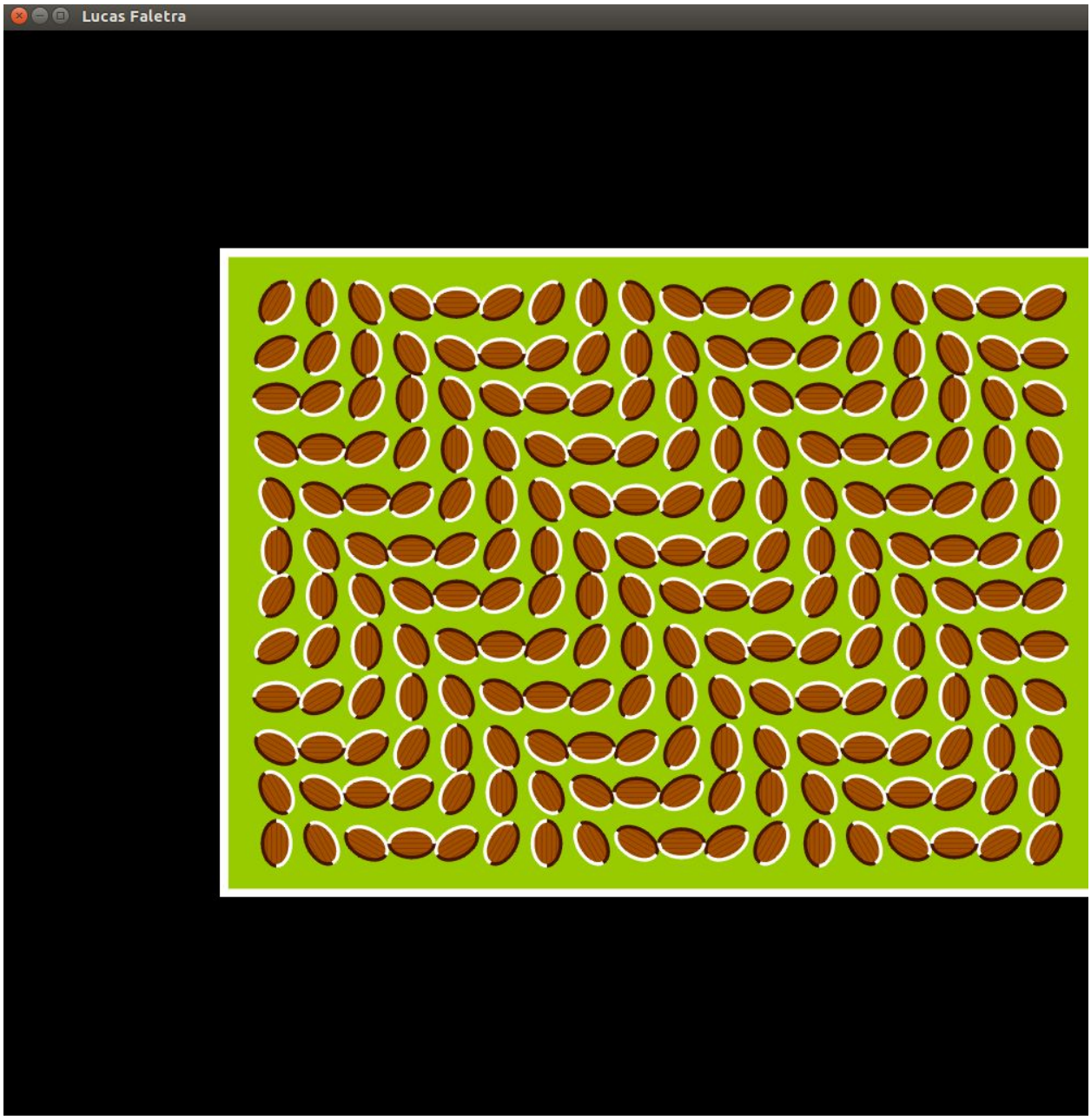
Key algorithms, data structures or OO design patterns:

This was the very first assignment for Computing IV, so there was not much use of objects or data structures, aside from the use of predefined classes in the SFML graphics library. The assignment was relatively simple and not very complex.

What I Learned:

In this assignment, I learned the basic syntax for using predefined classes in SFML to produce a sprite that responds to individual keystrokes. I learned the fundamentals of SFML and how the library's classes function to produce output a user can observe visually. I also learned the correct way to move an image using its coordinates, as well as the correct way to make that image respond to user input from the keyboard.

Evidence of Running Code:



```
/****************************************
Lucas Faletra
PS0 - SFML Hello world assignment
****************************************/

#include <SFML/Graphics.hpp>
int main(void){
   sf::RenderWindow window(sf::VideoMode(1000, 1000), "Lucas Faletra");


   int x, y = 100; //positions for sprite
   int limit = 10; //variable for frame limit
   window.setFramerateLimit(limit);

   while (window.isOpen())
       {
          sf::Event event;
          sf::Texture texture;
          texture.loadFromFile("sprite.png");
          sf::Sprite sprite;
          sprite.setTexture(texture);
          sprite.setPosition(x, y);

          while (window.pollEvent(event))
              {
                 if (event.type == sf::Event::Closed)
                     window.close();
                 else if(event.type == sf::Event::KeyPressed) /*if a key is pressed the sprite moves faster. Resets after reaching 100*/
                     {
                         if(limit > 100)
                           limit = 10;
                         else
                           limit += 10;
                         window.setFramerateLimit(limit);

                     }
              }
         x+=10;
         y+=10;
         if(x >= 750 || y >= 750)/*reset the x and y coordinates if they get too far out of range*/
              {
                     x = 100;
                     y = 100;
              }

         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Tilde)) /*if the ~ key is pressed the sprite is briefly rotated by 50 degrees*/
               sprite.setRotation(sprite.getRotation() + 50.0);
   window.clear();
   window.draw(sprite);
   window.display();
}

 return 0;
}
```

PS1a: Linear Feedback Shift Register

Assignment:

The assignment was to create a functional linear feedback shift register given a series of bits and an integer for the tap position. The second part of the assignment was to perform unit tests on the linear feedback shift register after finishing the implementation. I became familiar with the boost unit testing framework while completing this assignment. I also accomplished compiling all files into a single executable file using my own makefile.

I stored the register bits as a string and used string member functions to access bits. Then I converted the individual extracted characters to integers by subtracting 48 to get the value of the integer being represented and storing the result in integer variables. I did this because this was the easiest way I could think of accomplishing the task. The index of the tap is obtained by subtracting the initial tap value from the initial seed length, and then subtracting 1. I also gave the LFSR class a size data member to access for bounds checks, but it is not necessary because each member function could just use seed.length(). I just thought a size data member would make my code easier to read.

Key algorithms, data structures or OO design patterns:

The use of strings was necessary for reading in user input to determine the bits which the LFSR will be created with. Vectors could also be used in this case, but I found strings to be a fairly straightforward solution for this assignment. The basic use of classes, member functions, and private data members was also a requirement for this assignment. The algorithm for a linear feedback shift register was also necessary here. The algorithm for repeatedly shifting bits after performing an XOR operation on the initial seed and tap bit was crucial for functionality here. Unit testing was also a key concept for this assignment.

What I Learned:

During this assignment I learned how a linear feedback shift register operates through performing an XOR operation on the initial seed and tap bits. I was also introduced to the Boost unit testing framework. This allowed for me to test my code for the intended functionality.

Evidence of Running Code:

```
Terminal
File  Edit  View  Search  Terminal  Help
lfaletra@dan417-05:~/Downloads/ps1a_LucasFaletra/ps1a$ make
g++ -Wall -Werror -ansi -pedantic -c LFSR.hpp LFSR.cpp
g++ -Wall -Werror -ansi -pedantic -c test.cpp
g++ -Wall -Werror -ansi -pedantic -o ps1a LFSR.o test.o -lboost_unit_test_framew
ork
lfaletra@dan417-05:~/Downloads/ps1a_LucasFaletra/ps1a$ ./ps1a
Running 3 test cases...

*** No errors detected
lfaletra@dan417-05:~/Downloads/ps1a_LucasFaletra/ps1a$
```

```
CPP = g++ -Wall -Werror -ansi -pedantic
BOOST = -lboost_unit_test_framework
all: ps1a

ps1a: LFSR.o test.o
        $(CPP) -o ps1a LFSR.o test.o $(BOOST)

LFSR.o: LFSR.hpp LFSR.cpp
        $(CPP) -c LFSR.hpp LFSR.cpp
test.o: test.cpp
        $(CPP) -c test.cpp

clean:
        rm LFSR.o test.o ps1a LFSR.hpp.gch
```

```
/****************
Lucas Faletra
ps1a
***************/
#include <iostream>
#include <string>
#include "LFSR.hpp"

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

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);
}

BOOST_AUTO_TEST_CASE(initAt32Bits)
{
  //initialize a 32 bit LFSR with tap of 24
  //Perform a series of steps and check each return value
  LFSR test("00011100011100011100011100011100", 24);
  BOOST_REQUIRE(test.step() == 0);
  BOOST_REQUIRE(test.step() == 0);
  BOOST_REQUIRE(test.step() == 1);
  BOOST_REQUIRE(test.step() == 0);
  BOOST_REQUIRE(test.step() == 0);
  BOOST_REQUIRE(test.step() == 1);
  BOOST_REQUIRE(test.step() == 0);
  BOOST_REQUIRE(test.step() == 0);
  BOOST_REQUIRE(test.step() == 1);
  BOOST_REQUIRE(test.step() == 0);


  //call generate function on a 32 bit seed with tap of 30
  LFSR test2("00011100011100011100011100011100", 30);
  BOOST_REQUIRE(test2.generate(7) == 18);

}

BOOST_AUTO_TEST_CASE(tapOutOfBounds){
  /*this test checks that each function interperets the tap as out of bounds

 The if statement at the beginning of my step and generate functions perform
 no operations and returns -1 if the tap is not a valid index for the seed*/

  LFSR test("1010101", 32);//seed length is 7 bits, so 32 is not a valid tap value
  BOOST_REQUIRE(test.step() == -1);
  BOOST_REQUIRE(test.step() == -1);
```

```
BOOST_REQUIRE(test.generate(2) == -1);

LFSR test2("1010101", -12);//a negative tap value cannot be used
BOOST_REQUIRE(test2.step() == -1);
BOOST_REQUIRE(test2.step() == -1);
BOOST_REQUIRE(test2.generate(3) == -1);



}
```

```
/*****************
Lucas Faletra
ps1a
****************/
#include <string>
#include <iostream>
using namespace std;
class LFSR{
public:
  LFSR(string seed, int t);

  int step();

  int generate(int k);

  friend ostream& operator << (ostream& out, LFSR lfsr);

private:

  string bits;
  int tap;
  int size;
};


```

```
/*****************
Lucas Faletra
ps1a
*****************/
#include "LFSR.hpp"
#include <iostream>
#include <string>
using namespace std;

LFSR::LFSR(string seed, int t){
  this->bits = seed;
  this->size = seed.length();
  this->tap = this->size - t - 1;
}

int LFSR::step(){
  //first check if the tap is a valid index. Return -1 if it is not
  if(this->tap > this->size || this->tap < 0)
    {

      return -1;
    }
  int x, y, z;
  int i = 0;
  x = this->bits.at(0) - 48; //x is bit furthest to the left
  y = this->bits.at(this->tap) - 48; //y is the tap bit
  z = x ^ y;                 //XOR operation to get new rightmost bit
  while(i < this->size - 1)
    {
      this->bits.at(i) = this->bits.at(i+1); //shift all the bits to the left
      i++;
    }
  this->bits.at(this->size - 1) = z + 48;  //change the final bit to the result
  //of the XOR operation above

  return z;
}

int LFSR::generate(int k){
  //check if tap is a valid index. Return -1 if it is out of bounds
  if(this->tap > this->size || this->tap < 0)
    {
      return -1;
    }
  int count = 0;
  int i;
  for(i = 0; i < k; i++)
    {
      count *= 2;
      count += this->step();
    }
  return count;
}

ostream& operator << (ostream& out, LFSR lfsr){
  int n, i;
  for(i = lfsr.size - 1; i >= 0; i--)
    {
      n *= 2;
```

```
      n += lfsr.bits.at(i) - 48;
    }
  out << "Current value of register is: " << lfsr.bits << ' ' << n << endl;
  return out;
}

```

PS1b: PhotoMagic with Linear Feedback Shift Register

Assignment:

The goal of this assignment was to use the previously created LFSR class to encrypt and decrypt each individual pixel of a given image by changing the colors of the individual pixels. Running the program on a given image with a specified tap and seed bit should produce an output file that looks similar to static. Running the program on the output file produced previously, with the same tap and seed bits should reproduce the original image. I accomplished this by first creating the linear feedback shift register class, LFSR. Using this class, I was able to successfully change the rgb values for each pixel in the input image, based on the tap and seed bits given by the user. Unit tests on the LFSR class were also required to verify functionality.

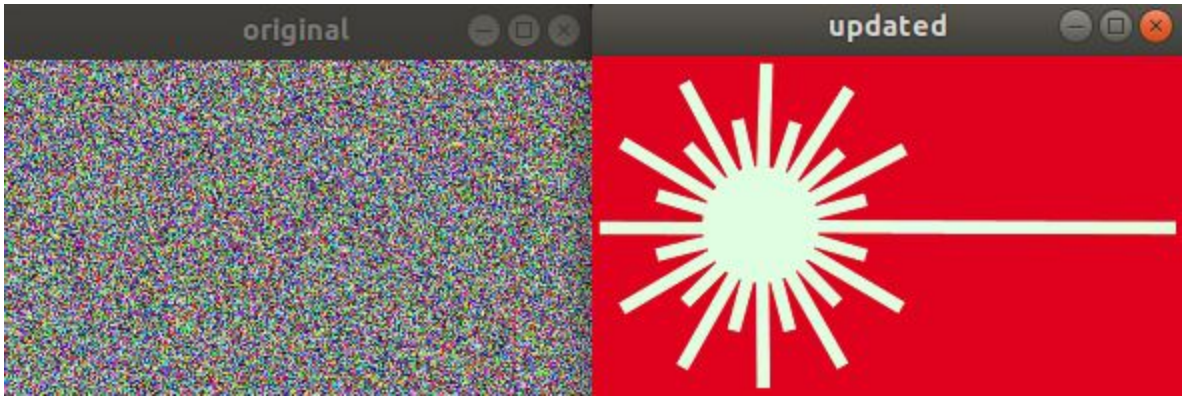
Key Algorithms, Data Structures, or OO concepts:

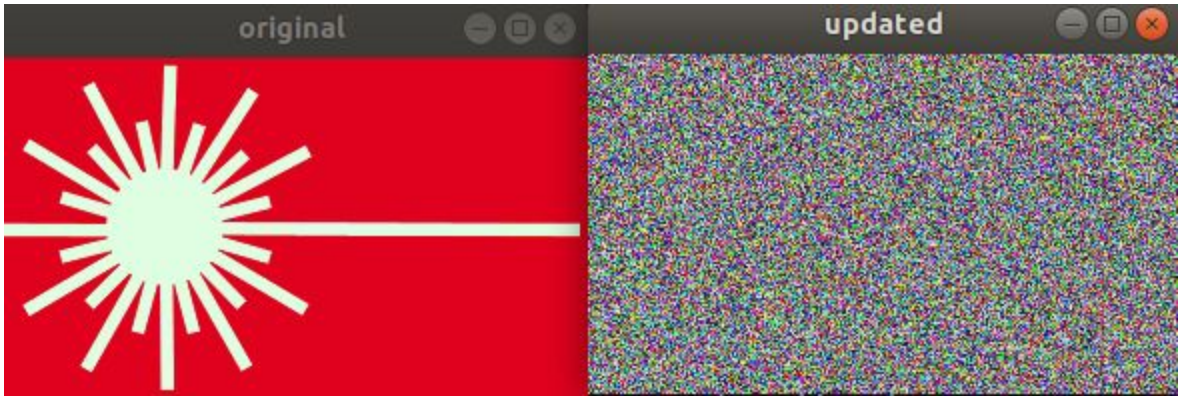
Strings and/or vectors were essential for this assignment in order to create the linear feedback shift register, depending on the implementation. I decided to use a string to store values into the linear feedback shift register because I thought that would be the easiest way to accomplish the assignment. Data encapsulation was also a key concept in this assignment. Data encapsulation allows for all essential information and values to be stored in the LFSR class and used later by the program when encrypting the image.

What I Learned:

In this assignment I learned how to create a linear feedback shift register with characteristics based on user input. I also learned how to use this linear feedback shift register and its operations to both encrypt and decrypt an image successfully. This assignment also taught me more about how SFML objects function on a fundamental level, as well as how to utilize their characteristics to produce interesting results. I also experienced unit testing in a C++ environment during this assignment for the first time using the boost unit-testing framework.

Evidence of Running Code:





```
CPP = g++ -Wall -Werror -ansi -pedantic
SFML = -lsfml-graphics -lsfml-window -lsfml-system
all: PhotoMagic

PhotoMagic: LFSR.o PhotoMagic.o
        $(CPP) -o PhotoMagic LFSR.o PhotoMagic.o $(SFML)

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

clean:
        rm LFSR.o PhotoMagic.o PhotoMagic LFSR.hpp.gch
```

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

int main()
{
  std::string source, destination, seed;
  int tap;
  std::cin >> source;
  std::cin >> destination;
  std::cin >> seed;
  std::cin >> tap;

  LFSR lfsr(seed, tap); //construct the LFSR with initial seed & tap

        sf::Image Source_png;
        if (!Source_png.loadFromFile(source))
                return -1;

        sf::Image Destination_png;
        if(!Destination_png.loadFromFile(destination))
          return -1;




        sf::Color p;
        sf::Color p2;
        for (unsigned int x = 0; x<Destination_png.getSize().x; x++) {
          for (unsigned int y = 0; y<Destination_png.getSize().y; y++) {
                       p = Destination_png.getPixel(x, y);
                       p2 = Source_png.getPixel(x, y);
                       p.r = p2.r ^ lfsr.generate(32);
                       p.g = p2.g ^ lfsr.generate(32);
                       p.b = p2.b ^ lfsr.generate(32);
                       Destination_png.setPixel(x, y, p);
               //set all pixels in destination image, given coordinates
               //loops through all pixels by accessing image dimensions

          }
        }

        sf::Vector2u size = Source_png.getSize();
        sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "original");

        sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "updated");

        sf::Texture texture_win1;
        texture_win1.loadFromImage(Source_png);

        sf::Texture texture_win2;
        texture_win2.loadFromImage(Destination_png);

        sf::Sprite sprite_win1;
        sprite_win1.setTexture(texture_win1);

        sf::Sprite sprite_win2;
        sprite_win2.setTexture(texture_win2);

```

```
        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(sprite_win1);
                window1.display();
                window2.clear();
                window2.draw(sprite_win2);
                window2.display();
        }
        if (!Destination_png.saveToFile(destination))
                return -1;

        return 0;
}
```

```
/*****************
Lucas Faletra
ps1b
***************/
#include <string>
#include <iostream>
using namespace std;
class LFSR{
public:
  LFSR(string seed, int t);

  int step();

  int generate(int k);

  friend ostream& operator << (ostream& out, LFSR lfsr);

private:

  string bits;
  int tap;
  int size;
};


```

```
/*****************
Lucas Faletra
ps1a
*****************/
#include "LFSR.hpp"
#include <iostream>
#include <string>
using namespace std;

LFSR::LFSR(string seed, int t){
  this->bits = seed;
  this->size = seed.length();
  this->tap = this->size - t - 1;
}

int LFSR::step(){
  //first check if the tap is a valid index. Return -1 if it is not
  if(this->tap > this->size || this->tap < 0)
    {

      return -1;
    }
  int x, y, z;
  int i = 0;
  x = this->bits.at(0) - 48; //x is bit furthest to the left
  y = this->bits.at(this->tap) - 48; //y is the tap bit
  z = x ^ y;                 //XOR operation to get new rightmost bit
  while(i < this->size - 1)
    {
      this->bits.at(i) = this->bits.at(i+1); //shift all the bits to the left
      i++;
    }
  this->bits.at(this->size - 1) = z + 48;  //change the final bit to the result
  //of the XOR operation above

  return z;
}

int LFSR::generate(int k){
  //check if tap is a valid index. Return -1 if it is out of bounds
  if(this->tap > this->size || this->tap < 0)
    {
      return -1;
    }
  int count = 0;
  int i;
  for(i = 0; i < k; i++)
    {
      count *= 2;
      count += this->step();
    }
  return count;
}

ostream& operator << (ostream& out, LFSR lfsr){
  int n, i;
  for(i = lfsr.size - 1; i >= 0; i--)
    {
      n *= 2;
```

```
      n += lfsr.bits.at(i) - 48;
    }
  out << "Current value of register is: " << lfsr.bits << ' ' << n << endl;
  return out;
}

```

PS2: Recursive Graphics (Pythagoras Tree)

Assignment:

The assignment was to draw a pythagoras tree by making a class which can recursively draw itself to the screen. The dimensions and depth of recursion are determined by the two parameters the user enters. The first parameter from the user determines the length of the sides of the base square, and window has a width of 6 times this length and a height of 4 times the length of the side of the base square. The user also enters a second parameter which determines the depth of recursion(how many times the tree will branch out). I implemented this by creating a Ptree class with two ConvexShape objects from the SFML library. One object is a square and the other is a triangle. Using the public member functions I created, I can use the x and y coordinates of the current square and triangle to determine the coordinates of the next square and triangle. Once the program gets the coordinates of the next two shapes, a new Ptree is created using the second constructor. The depth of recursion also decrements each time a new Ptree is created. As the depth decrements, all objects are drawn recursively until the depth reaches 0.

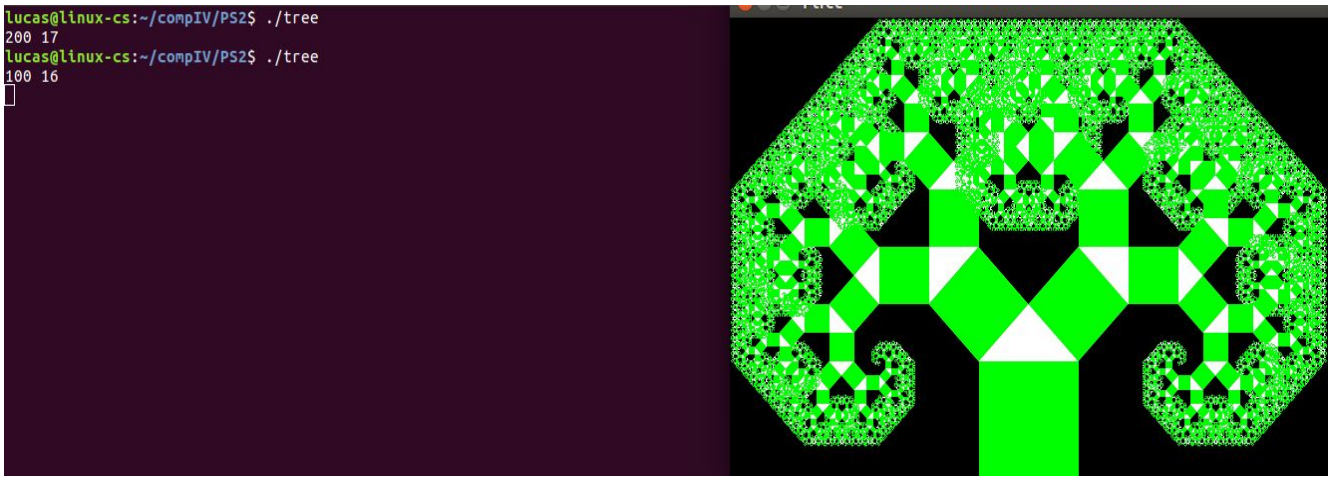
Key Algorithms, Data Structures, or OO concepts:

One key algorithm that I used during this assignment was the preorder traversal of a binary search tree. My draw function is very similar to this algorithm since it first evaluates and draws the current Ptree, then proceeds to the left, then to the right after finishing the left in a recursive pattern. A data structure which was very helpful in this assignment was a vector. Since coordinates are stored in vectors, the easiest way for me to store multiple coordinates was to create a vector containing the sf::Vector2f type.

What I Learned:

In this assignment I learned how to recursively create shapes using the SFML library. I also learned about fractals and how to use them to create interesting objects. Another thing I learned in this assignment was that certain algorithms are easily transferable to many different kinds of projects, as was the case of using a tree traversal algorithm in my draw function.

Evidence of Running Code:


lucas@linux-cs:~/compIV/PS2$ ./tree
200 17
lucas@linux-cs:~/compIV/PS2$ ./tree
100 16

```
CPP = g++ -O3 -Wall -Werror -pedantic
SFML = -lsfml-graphics -lsfml-window -lsfml-system

all: tree

tree: main.o Ptree.o
        $(CPP) -o tree main.o Ptree.o $(SFML)
main.o: main.cpp
        $(CPP) -c main.cpp

Ptree.o: Ptree.hpp Ptree.cpp
        $(CPP) -c Ptree.hpp Ptree.cpp
clean:
        rm *~ *.o tree Ptree.hpp.gch
```

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <SFML/System.hpp>
#include <vector>
#include "Ptree.hpp"
using namespace std;


int main(){


  double L;
  int N;
  cin >> L;
  cin >> N;
  Ptree tree(L, N);
  sf::RenderWindow window(sf::VideoMode((6 * L), (4 * L)), "Ptree");


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


    }
  return 0;
}
```

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <SFML/System.hpp>
#include <vector>
using namespace std;

class Ptree:public sf::Drawable{
public:
Ptree(double L, int N);
Ptree(vector<sf::Vector2f> previous, int N, char lr_val);
int getDepth() const {return depth;}; //basic accessor for main function
  vector<sf::Vector2f> calculate_r_points(vector<sf::Vector2f> previous) const;
  vector<sf::Vector2f> calculate_l_points(vector<sf::Vector2f> previous) const;

  void printValues();

  vector<sf::Vector2f> getTriangle()const;
  void draw(sf::RenderTarget& target, sf::RenderStates states) const;

private:
  double start;
  int depth;


  sf::ConvexShape base;         //base is only utilized if first constructor called
  sf::ConvexShape triangle;     //after base called, init remaining shapes


  vector<sf::Vector2f> points;


};
```

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <SFML/System.hpp>
#include <vector>
#include "Ptree.hpp"
using namespace std;

//this function is for accessing the points of the triangle
//I used this to get a parameter for initializing a new Ptree object
vector<sf::Vector2f> Ptree::getTriangle() const{
  vector<sf::Vector2f> temp;
  temp.push_back(this->triangle.getPoint(0));
  temp.push_back(this->triangle.getPoint(1));
  temp.push_back(this->triangle.getPoint(2));
  return temp;
}


//initial constructor. Sets the length of the base square and
//the points of both the first square and first triangle based on 2 parameters
Ptree::Ptree(double L, int N)
{

    this->start = L;
    this->depth = N;

    this->base.setPointCount(4); //set the coordinates of the base square
    this->base.setPoint(0, sf::Vector2f(3 * L - (L/2.0), 4 * L));//
    this->base.setPoint(1, sf::Vector2f(3 * L + (L/2.0), 4 * L));
    this->base.setPoint(2, sf::Vector2f(3 * L + (L/2.0), (4 * L) - L));
    this->base.setPoint(3, sf::Vector2f(3 * L - (L/2.0), (4 * L) - L));//

    this->base.setFillColor(sf::Color::Green);

    this->triangle.setPointCount(3);//set coordinates of first triangle
    sf::Vector2f p0 = this->base.getPoint(3);
    sf::Vector2f p1 = this->base.getPoint(2);
    sf::Vector2f p2;
    p2.x = (3 * L)/1.0;
    p2.y = (4 * L) - (L * 1.5);


    this->triangle.setPoint(0, p0);
    this->triangle.setPoint(1, p1);
    this->triangle.setPoint(2, p2);


    this->points.push_back(triangle.getPoint(0));
    this->points.push_back(triangle.getPoint(1));
    this->points.push_back(triangle.getPoint(2));

    this->triangle.setFillColor(sf::Color::White);
}

/**************************************************
This is a second constructor I use for creating new objects based on the
initial Ptree object created from user input. The constructor uses a vector of three points from the previous object's triangle to create a new square, sets
a new recursion depth, and uses lr_val to determine whether the new square w
```

```
ill
be oriented to the left or right.
*****************************************************/
Ptree::Ptree(vector<sf::Vector2f> previous, int N, char lr_val){
 vector<sf::Vector2f> points;

 points.reserve(3);
 sf::Vector2f p3;

 this->depth = N;
 this->base.setPointCount(4);
 this->triangle.setPointCount(3);
if(lr_val == 'l')
  {
    this->base.setPoint(0, previous.at(0));
    this->base.setPoint(1, previous.at(2));

    this->base.setPoint(2, this->calculate_l_points(previous).at(0));
    this->base.setPoint(3, this->calculate_l_points(previous).at(1));
    this->triangle.setPoint(0, this->base.getPoint(3));
    this->triangle.setPoint(1, this->base.getPoint(2));

    p3.x = this->base.getPoint(3).x + ((this->base.getPoint(2).x -  this->base.getPoint(0).x) * 0.5);
    p3.y = this->base.getPoint(2).y + ((this->base.getPoint(3).y - this->base.getPoint(1).y) * 0.5);
    this->triangle.setPoint(2, p3);
  }


 else
   {
     this->base.setPoint(0, previous.at(2));
     this->base.setPoint(1, previous.at(1));

     this->base.setPoint(2, this->calculate_r_points(previous).at(0));
     this->base.setPoint(3, this->calculate_r_points(previous).at(1));

     this->triangle.setPoint(0, this->base.getPoint(3));
     this->triangle.setPoint(1, this->base.getPoint(2));

     p3.x = this->base.getPoint(3).x - ((this->base.getPoint(0).x - this->base.getPoint(2).x) * 0.5);
     p3.y = this->base.getPoint(2).y - ((this->base.getPoint(1).y - this->base.getPoint(3).y) * 0.5);
     this->triangle.setPoint(2, p3);
   }

 this->base.setFillColor(sf::Color::Green);
}

/*****************************************************
This member function is used in the second constructor mentioned above. It
takes the vector of 3 coordinates passed to the constructor and
calculates the coordinates of the next left-oriented square
*****************************************************/
vector<sf::Vector2f> Ptree::calculate_l_points(vector<sf::Vector2f> previous) const
{
  sf::Vector2f p2;
  sf::Vector2f p3;
```

```
  vector<sf::Vector2f> ret_points;
  ret_points.clear();
  ret_points.reserve(2);
  p2.x = previous.at(2).x + (previous.at(2).x - previous.at(1).x);
  p2.y = previous.at(2).y + (previous.at(2).y - previous.at(1).y);

  p3.x = previous.at(0).x + (previous.at(2).x - previous.at(1).x);
  p3.y = previous.at(0).y + (previous.at(2).y - previous.at(1).y);
  ret_points.push_back(p2);
  ret_points.push_back(p3);


  return ret_points;

}

/*****************************************************
The calculate_r_points member function is almost identical to the calculate_l_points member function, but has a different set of formulas for getting the location of the next right-oriented square. I made a separate member function for left
and right because I thought it would be easier this way.
*****************************************************/
vector<sf::Vector2f> Ptree::calculate_r_points(vector<sf::Vector2f> previous) const
{
  sf::Vector2f p2;
  sf::Vector2f p3;
  vector<sf::Vector2f> ret_points;
  ret_points.clear();
  ret_points.reserve(2);

  //calculate point 2 and 3 of the next base based on previous triangle coordinates
  p2.x = previous.at(1).x + (previous.at(2).x - previous.at(0).x);
  p2.y = previous.at(1).y + (previous.at(2).y - previous.at(0).y);

  p3.x = previous.at(2).x + (previous.at(2).x - previous.at(0).x);
  p3.y = previous.at(2).y + (previous.at(2).y - previous.at(0).y);

  ret_points.push_back(p2);
  ret_points.push_back(p3);

  return ret_points;
  //returns vector of points 2 and 3 for next square to be drawn for right


}

/*****************************************************
I decided to make redefine the draw function to be recursive. It uses an algorithm very similar to a preorder tree traversal to draw all the objects
*****************************************************/
void Ptree::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
  vector<sf::Vector2f> tri;

  if(this->depth >= 1)
    {
      tri = this->getTriangle();
      target.draw(this->base);
      target.draw(this->triangle, states);
```

```
      Ptree left(tri, this->depth - 1, 'l');
      target.draw(left, states);
      Ptree right(tri, this->depth - 1, 'r');
      target.draw(right, states);

    }
  else
    {
      target.draw(this->base);
      //return;
    }

}

/**********************************************
I made this function for testing purposes only. It is not used anywhere in the main program. All it does is print the values of the base and triangle data members
**********************************************/
void Ptree::printValues(){
  cout << "BASE:" << endl;
  cout << "POINT 0: " << this->base.getPoint(0).x << ' ' << this->base.getPoint(0).y << endl;
  cout << "POINT 1: " << this->base.getPoint(1).x << ' ' << this->base.getPoint(1).y << endl;
  cout << "POINT 2: " << this->base.getPoint(2).x << ' ' << this->base.getPoint(2).y << endl;
  cout << "POINT 3: " << this->base.getPoint(3).x << ' ' << this->base.getPoint(3).y << endl;

  cout << "TRIANGLE: " << endl;
   cout << "POINT 0: " << this->triangle.getPoint(0).x << ' ' << this->triangle.getPoint(0).y << endl;
  cout << "POINT 1: " << this->triangle.getPoint(1).x << ' ' << this->triangle.getPoint(1).y << endl;
  cout << "POINT 2: " << this->triangle.getPoint(2).x << ' ' << this->triangle.getPoint(2).y << endl;
}

```

PS3a: N-Body Simulation with Static Universe

Assignment:

The goal of this assignment was to display a static universe using SFML, provided image files for planets, and provided text files which contain the x-y coordinates of each planet. The assignment was to only display the first four planets, along with the sun in the center of the window. First, I needed to create a class called Body which could store the essential information for a single planet(in this case, the x and y coordinates). The constructor initializes the Body object from a given universe radius, x and y coordinates, image file, and name. The draw function for Body is redefined to make the object drawable. Finally, the istream operator >> is overloaded so that the object has the ability to seamlessly read in data and set its private data members.

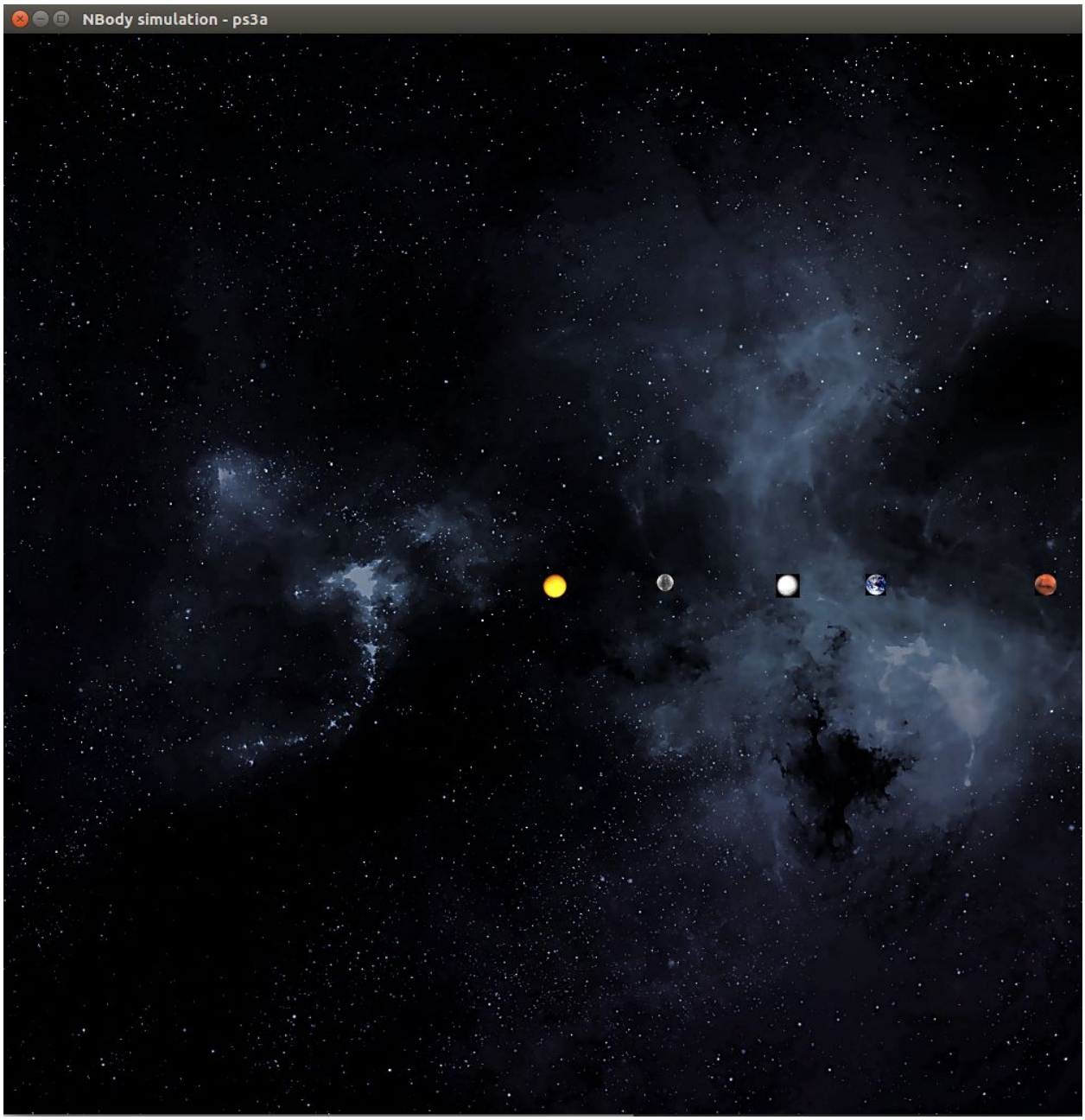
Key Data Structures, Algorithms and OO Design Patterns:

Vectors were essential for this assignment. One of the easiest ways to hold an arbitrary number of smart pointers associated with body objects is to place them into a vector. Also the assignment instructions said to use a vector for holding the objects. Strings were also useful for me. I used strings to parse the information contained in the planets.txt file and store that information as numeric values. Smart pointers were also central to this assignment. I never knew about smart pointers before this assignment, but they are actually very useful. Smart pointers allow for the user to reduce the possibility for a potential memory leak.

What I Learned:

During this assignment, I learned what smart pointers are. Smart pointers are very useful for a number of reasons but the primary concept is the fact that they eliminate the possibility of a memory leak, making them an essential part of any program using arbitrary pointers or memory allocation.

Evidence of Running Code:



```
CPP = g++ -O3 -Wall -Werror -std=c++14
SFML = -lsfml-graphics -lsfml-window -lsfml-system

all: NBody

NBody: main.o Body.o
        $(CPP) -o NBody main.o Body.o $(SFML)
main.o: main.cpp
        $(CPP) -c main.cpp

Body.o: Body.hpp Body.cpp
        $(CPP) -c Body.hpp Body.cpp
clean:
        rm *~ *.o NBody Body.hpp.gch
```

```
#include <iostream>
#include <vector>
#include <string>
#include "Body.hpp"
using namespace std;
int main(){

  int i = 0;
  int num_planets;
  string radius_string;
  double decimal;
  double radius;

  sf::Texture texture;
  texture.loadFromFile("background.png");
  sf::Sprite sprite;
  sprite.setTexture(texture);

  vector <unique_ptr<Body>> pointers;

  cin >> num_planets;
  cin >> radius_string;

  decimal = stod(radius_string.substr(0, radius_string.find("+")));
  radius = decimal * 200;

  sprite.setPosition(0, 0);

  for(i = 0; i < num_planets; i++)
      pointers.push_back(make_unique<Body>(radius));
  for(i = 0; i < num_planets; i++)
      cin >> *(pointers.at(i));

  sf::RenderWindow window(sf::VideoMode((radius * 2), (radius * 2)), "NBody simulation - ps3a");
  while(window.isOpen()){
    sf::Event event;
    while(window.pollEvent(event))
      {
        if(event.type == sf::Event::Closed)
          window.close();
      }

  window.clear();
  window.draw(sprite);
  for(i = 0; i < num_planets; i++)
    window.draw(*(pointers.at(i)));

  window.display();
  }
  return 0;
}
```

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

using namespace std;

class Body:public sf::Drawable{
 public:
//constructor needs to load image into texture, then load texture into sprit
e
  Body(double rad){this->U_center = rad;};
  Body(double radius_point, string x, string y, string velx, string vely, st
ring m, string file_name);

 void draw(sf::RenderTarget& target, sf::RenderStates state) const;
  friend istream &operator>>( istream &input, Body &B);
 private:
  sf::Vector2f center;
  string x, y, mass, velx, vely;
  double U_center;
  string file_name;
  sf::Texture texture;
  sf::Sprite sprite;

};
```

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

using namespace std;

Body::Body(double radius_point, string x, string y, string velx, string vely, string m, string file_name){
  this->center.x = radius_point + (stod(x.substr(0, x.find("e"))) * 200);
  this->center.y = radius_point + (stod(y.substr(0, y.find("e"))) * 200);


  //only x and y coordinates matter in this assignment, so I stored the rest of the info in strings
  this->velx = velx;
  this->vely = vely;
  this->mass = m;
  this->U_center = radius_point;


  this->texture.loadFromFile(file_name);
  this->sprite.setTexture(this->texture);
  this->sprite.setPosition(this->center);

}

//draw the sprite object
void Body::draw(sf::RenderTarget& target, sf::RenderStates states) const{
  target.draw(this->sprite, states);
}

//read in parameters and set the object’s private data members
istream &operator>>(istream &input, Body &B){
  string tempy;
  string tempx;
  input >> tempx >> tempy >> B.velx >> B.vely >> B.mass >> B.file_name;

  //use the radius as an origin for x and y coordinates. Parse the number in the planets.txt file and scale it in a similar way to radius
  B.center.x = B.U_center + ((stod(tempx.substr(0, tempx.find("e")))) / ((11.0 - (stod(tempx.substr(tempx.find("e") + 1, tempx.length()))) * 10 + 1) * 200);
  B.center.y = B.U_center + (stod(tempy.substr(0, tempy.find("e"))) * 200);


  //set up the image that will be associated with the object
  B.texture.loadFromFile(B.file_name);
  B.sprite.setTexture(B.texture);
  B.sprite.setPosition(B.center);
  return input;
}
```

PS3b: N-Body Simulation with Orbits

Assignment:

The assignment was to create a simulation of the first four planets of the solar system and their orbits around the sun using physics equations based on forces acting on different particles. The necessary information was read from a text file provided which lists aspect of each planet such as the x and y coordinates, the x and y components of a planet's velocity, and each planet's mass. The goal of this assignment was to parse this information correctly and then create the simulation. I was able to display a static solar system by creating a vector of smart pointers in my main program. Each smart pointer contains exactly one planet with all of its characteristics, defined as a Body class type. Later in the main program, each planet is drawn to an SFML window in a loop.

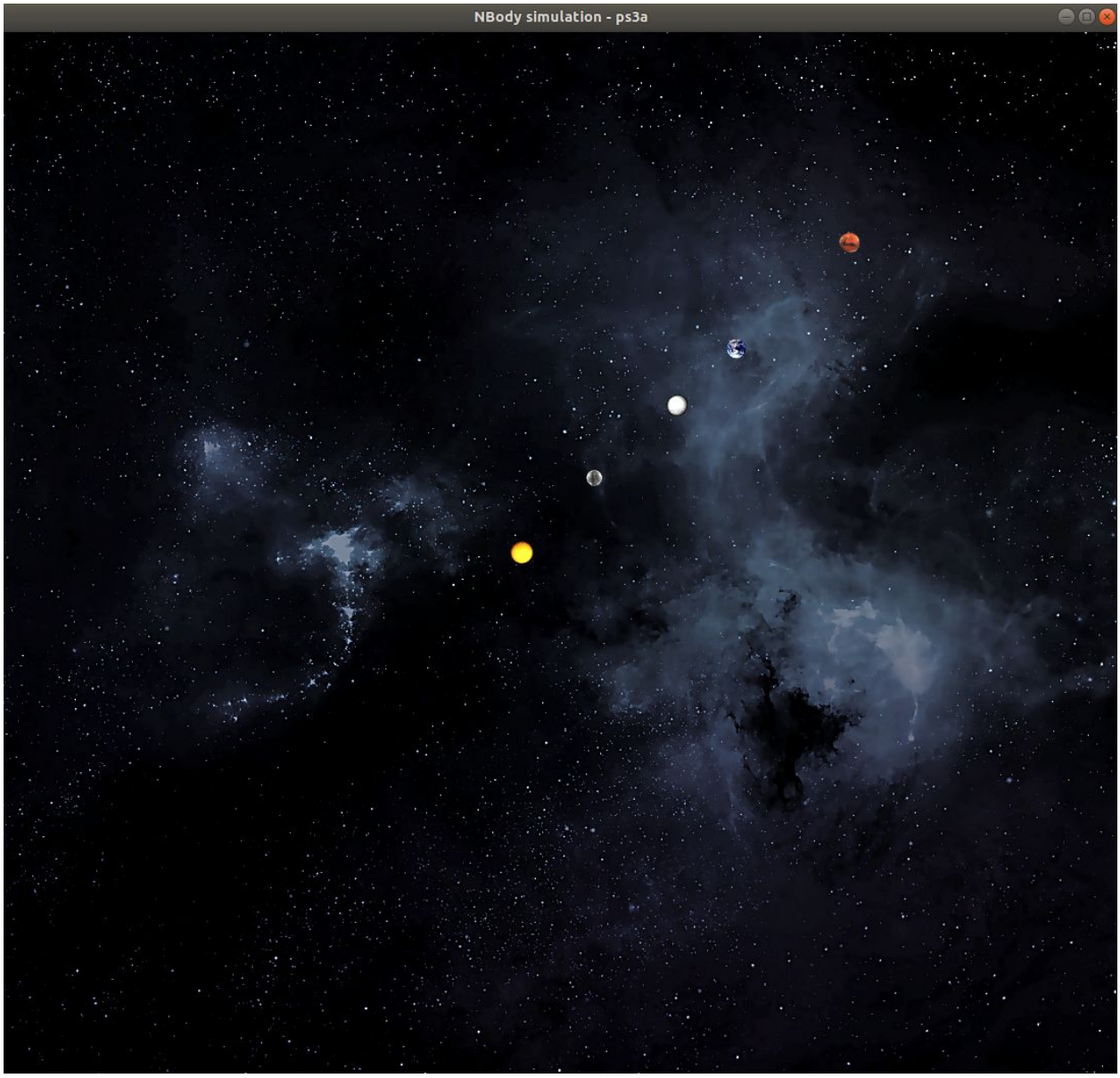
Key algorithms, data structures or OO design patterns:

Smart pointers, vectors and strings were central to this assignment. Smart pointers can be used to more efficiently store the addresses of the Body objects corresponding to the planets. A vector can be used to store all the Body smart pointers. Strings can be used to parse the user input from the command line. Classes were also useful for encapsulating each planet as a separate object.

What I Learned:

During this assignment, I learned how to properly utilize smart pointers to safely reference a class, without risk of a potential memory leak. I also became more familiar with the SFML library and properties of sprites and images in this library.

Evidence of Running Code:



Parts That did not Work:

I could not figure out how to properly use the provided equations to create the planets' orbits, so I used the equation of a circle in polar coordinates for each planet and attempted to make them move at different speeds. The planets are drawn properly and they move in a counterclockwise direction, but they do not follow their intended orbits because of my implementation.

```
CPP = g++ -O3 -Wall -Werror -std=c++14
SFML = -lsfml-graphics -lsfml-window -lsfml-system

all: NBody

NBody: main.o Body.o
        $(CPP) -o NBody main.o Body.o $(SFML)
main.o: main.cpp
        $(CPP) -c main.cpp

Body.o: Body.hpp Body.cpp
        $(CPP) -c Body.hpp Body.cpp
clean:
        rm *~ *.o NBody Body.hpp.gch
```

```
#include <iostream>
#include <vector>
#include <string>
#include "Body.hpp"
using namespace std;
int main(int argc, char* argv[]){

  int i = 0;
  int num_planets;
  string radius_string;
  double decimal;
  double radius;
  double t, delta_t;
  double time = 0;

  sf::Texture texture;
  texture.loadFromFile("background.png");
  sf::Sprite sprite;
  sprite.setTexture(texture);

  vector <unique_ptr<Body>> pointers;

  string a = argv[1];
  string b = argv[2];

  t = stod(a.substr(0, a.length()));
  delta_t = stod(b.substr(0, b.length()));

  cin >> num_planets;
  cin >> radius_string;

  decimal = stod(radius_string.substr(0, radius_string.find("+")));
  radius = decimal * 200;

  sprite.setPosition(0, 0);


  for(i = 0; i < num_planets; i++)
      pointers.push_back(make_unique<Body>(radius));
  for(i = 0; i < num_planets; i++)
      cin >> *(pointers.at(i));
  for(i = 0; i < num_planets; i++)
    {
      if(i != 3)
        pointers.at(i)->calc_forces(*(pointers.at(3)));


    }

  sf::RenderWindow window(sf::VideoMode((radius * 2.2), (radius * 2.2)), "NBody simulation - ps3a");
  window.setFramerateLimit(10);
  while(window.isOpen()){
    sf::Event event;
    while(window.pollEvent(event))
      {
        if(event.type == sf::Event::Closed)
          window.close();
      }

  window.clear();
```

```
  window.draw(sprite);
  //test of new formula on venus

  if(time < t)
    {

      window.draw(*(pointers.at(3)));

      //pointers.at(2)->calc_forces(*(pointers.at(3)));
      //pointers.at(4)->calc_forces(*(pointers.at(3)));
      //pointers.at(0)->calc_forces(*(pointers.at(3)));
      //pointers.at(1)->calc_forces(*(pointers.at(3)));

      for(i = 0; i < num_planets; i++)
        {
          pointers.at(i)->step(*(pointers.at(3)), delta_t);
          window.draw(*(pointers.at(i)));
          }
      time+=delta_t;
      window.display();
  }

  else
    {
      for(i = 0; i < num_planets; i++)
        {
          window.draw(*(pointers.at(i)));
        }

      window.display();
      }
    }
  return 0;
}
```

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

using namespace std;

class Body:public sf::Drawable{
 public:
//constructor needs to load image into texture, then load texture into sprit
e
  Body(double rad){this->U_center = rad;};
  Body(double radius_point, string x, string y, string velx, string vely, st
ring m, string file_name);



  //basic accessor functions for x and y components and mass
  double get_x(){return this->center.x;};
  double get_y(){return this->center.y;};
  double get_mass(){return this->mass;};
  sf::Vector2f get_force(){return this->forces;};
  void calc_forces(Body B);

  void step(Body B, double time);




  void draw(sf::RenderTarget& target, sf::RenderStates state) const;
  friend istream &operator>>( istream &input, Body &B);

private:
  sf::Vector2f center;
  sf::Vector2f forces;

  double angle = 0;
  string x, y, m, velx, vely;

  double U_center;

  double mass, velocity_x, velocity_y;

  string file_name;

  sf::Texture texture;

  sf::Sprite sprite;

};
```

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

using namespace std;

Body::Body(double radius_point, string x, string y, string velx, string vely
, string m, string file_name){
  this->center.x = radius_point + (stod(x.substr(0, x.find("e"))) * 200);
  this->center.y = radius_point + (stod(y.substr(0, y.find("e"))) * 200);


  this->velx = velx;
  this->vely = vely;
  this->m = m;
  this->U_center = radius_point;


  this->texture.loadFromFile(file_name);
  this->sprite.setTexture(this->texture);
  this->sprite.setPosition(this->center);

}

void Body::calc_forces(Body B)
{

  if(this->angle == 2)
    this->angle = 0;

  if(this->angle == 0)
    this->velocity_x = (this->center.x - this->U_center) / 200;

  this->center.x -= this->velocity_x * (sin(this->angle));
  this->center.y -= this->velocity_x * (cos(this->angle));

  this->sprite.setPosition(this->center.x, this->center.y);



  this->angle+= (1/(180.0  + (this->mass/10)));
}

void Body::step(Body B, double time){
  this->calc_forces(B);
}

void Body::draw(sf::RenderTarget& target, sf::RenderStates states) const{
  target.draw(this->sprite, states);
}

istream &operator>>(istream &input, Body &B){
  string tempy;
  string tempx;
  string velx, vely, m;
  input >> tempx >> tempy >> velx >> vely >> m >> B.file_name;

```

```
   B.velx = velx;
   B.vely = vely;
   B.m = m;

   B.center.x = B.U_center + ((stod(tempx.substr(0, tempx.find("e")))) / ((11
.0 - (stod(tempx.substr(tempx.find("e") + 1, tempx.length()))))) * 10 + 1) * 200);

  B.center.y = B.U_center + ((stod(tempy.substr(0, tempy.find("e")))) / ((11.
0 - (stod(tempy.substr(tempy.find("e") + 1, tempy.length()))))) * 10 + 1) * 200);

   B.velocity_x = ((stod(velx.substr(0, velx.find("e")))) / ((11.0 - (stod(ve
lx.substr(velx.find("e") + 1, velx.length()))))) * 10 + 1) * 200);


   B.velocity_y = ((stod(vely.substr(0, vely.find("e")))) / ((11.0 - (stod(ve
ly.substr(vely.find("e") + 1, vely.length()))))) * 10 + 1) * 200);


   B.mass =  10 * ((stod(m.substr(0, m.find("e")))) / ((11.0 - (stod(m.substr
(m.find("e") + 1, m.length() - 1)))) * 10 + 1) * 200);
   B.texture.loadFromFile(B.file_name);
   B.sprite.setTexture(B.texture);
   B.sprite.setPosition(B.center);

   B.forces.x = 0;
   B.forces.y = 0;
   return input;
 }
```

PS4: DNA Sequence Alignment

Assignment:

The goal of this assignment was to find the optimal alignment for any two given strings resembling DNA sequences, then compute the appropriate edit distance. This was accomplished by creating a two dimensional matrix based on the lengths of the input strings. The bottom row of the matrix is filled with values from 0 to 2N, incrementing by two each index, where N is the length of the second string. The rightmost column is filled with values from 0 to 2M, where M is the length of the first string input to the program. At this point, each index of the matrix is filled with values equivalent to the minimum value of one of three values. These values are opt[i+1][j+1] + 0/1, opt[i+1][j] + 2, opt[i][j+1] + 2, where i and j are indexes of the matrix. After filling the matrix with values, the edit distance is computed by traversing the matrix and applying penalties when necessary. This project also required analysis of time complexity through the use of valgrind.

Key Algorithms, Data Structures, or OO concepts:

This assignment required the use of strings and vectors. The two DNA sequences were read in from the input file and stored as strings by the program. Two vectors of integers are necessary for storing the matrix and computing the appropriate edit distance. The given algorithm for computing edit distance was also very important for this assignment. Using a basic iterative or even a recursive implementation for this assignment would cause numerous issues in terms of memory and performance. Time complexities were another key concept in the design of this project.

What I Learned:

During this project, I gained an in depth understanding of the importance of time complexities as well as a fundamental understanding of the algorithm for computing the edit distance of two DNA sequences. I also learned how to use the more complex features of Valgrind to analyze the time complexity and memory usage of a program visually.

Evidence of Running Code:

```
Terminal
File  Edit  View  Search  Terminal  Help
lfaletra@dan417-05:~/compIV/ps4$ ./ED <endgaps7.txt
Edit Distance is: 4
a - 2
t t 0
a a 0
t t 0
t t 0
a a 0
t t 0
- a 2
Execution time is 3.9e-05 seconds
lfaletra@dan417-05:~/compIV/ps4$
```

```
C++ = g++ -std=c++14 -O3
SFML = -lsfml-system -lsfml-graphics
all: ED

ED: main.o ED.o
        $(C++) -o ED main.o ED.o $(SFML)

main.o: main.cpp
        $(C++) -c main.cpp

ED.o: ED.cpp ED.hpp
        $(C++) -c ED.cpp ED.hpp

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

```
#include "ED.hpp"
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <string>
#include <vector>
#include <iostream>
int main(){

  sf::Clock clock;
  sf::Time t;

  std::string str1, str2;

  std::cin >> str1;

  std::cin >> str2;

  ED test(str1, str2);

  std::cout << test.Allignment();

  std::cout << test.get_distance() << std::endl;
  t = clock.getElapsedTime();
  std::cout << "Execution time is " << t.asSeconds() << " seconds" << std::endl;
  return 0;
}
```

```
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <string>
#include <vector>
class ED{
public:
  ED(std::string str1, std::string str2);
  int penalty(char a, char b);
  int min(int a, int b, int c);
  int OptDistance();
  int get_distance(){return this->distance;};
  std::string Allignment();

private:
  std::string first, second;
  std::vector<std::vector<int>> numbers;
  int M, N;
  int distance;

};
```

```
 1: #include "ED.hpp"
 2: #include <SFML/System.hpp>
 3: #include <SFML/Graphics.hpp>
 4: #include <string>
 5: #include <vector>
 6: #include <iostream>
 7: #include <algorithm>
 8:
 9:
10:
11: ED::ED(std::string str1, std::string str2){
12:
13:   this->first = str1;
14:
15:   this->second = str2;
16:
17:   int i = 0;
18:
19:   int j = 0;
20:
21:   int k = 0;
22:
23:   this->first += "-";
24:
25:   this->second += "-";
26:
27:   int collumn_size = str1.length();
28:
29:   int n_rows = str2.length();
30:
31:   this->M = collumn_size;
32:
33:   this->N = n_rows;
34:
35:   std::vector<std::vector<int>> numbers(M + 1);
36:
37:   for(i = 0; i <= this->N; i+=2)
38:
39:     {
40:
41:       numbers.at(M).push_back(this->N - (i));
42:
43:     }
44:
45:
46:   for(i = 0; i <= M; i++)
47:     for(j = 0; j <= N; j++)
48:       numbers.at(i).push_back(0);
49:
50:  for(i = (this->M); i >= 0; i--)
51:
52:     {
53:
54:       numbers.at(i).back() = (2 * (this->M - i));
55:
56:     }
57:
58:
59:
60:   this->numbers = numbers;
61:   }
```

```
int ED::penalty(char a, char b){

  if(a == b)

    return 0;

  else

    return 1;

}

int ED::min(int a, int b, int c){

  int min;

  min = std::min(a, b);

  min = std::min(min, c);

  return min;

}

int ED::OptDistance(){

  //populate the matrix

  //traverse it to find opt distance


  int i, j;

  int val;

  int distance = 0;

  //std::vector<std::vector<int>> numbers = this->numbers;

  for(i = ((M)-1); i >= 0; i--)

    {

      for(j = ((N)-1); j >= 0; j--)

        {

          val = this->penalty(this->first.at(i), this->second.at(j));

          numbers[i][j] = this->min((numbers.at(i+1).at(j+1)) + val, (numbers.at(i+1).at(j)) + 2, (numbers.at(i).at(j+1)) + 2);

        }



    }

  this->numbers = numbers;

```

```
  return numbers[0][0];

}

std::string ED::Allignment(){

  std::string ret;

  int i = 0;

  int j = 0;
  int distance = this->OptDistance();


  while(i < M && j < N)

    {

      if(this->numbers[i][j] == this->numbers[i+1][j+1] + this->penalty(this
->first.at(i), this->second.at(j)))

          {
              ret += (this->second.at(j));

              ret += (" ");

              ret += (this->first.at(i));

              if(this->penalty(this->first.at(i), this->second.at(j)) == 0)
                  ret += (" 0  \n");

              else
                  ret += (" 1  \n");

              i+=1;

              j+=1;

           }



      else if(this->numbers[i][j] == this->numbers[i+1][j] + 2)


         {

          ret += "- ";

          ret += this->first.at(i);

          ret += " 2  \n";

          i+=1;

         }


       else if(this->numbers[i][j] == this->numbers[i][j+1] + 2)
```

```
         {

          ret += this->second.at(j);

          ret += (" - 2   \n");

          j+=1;
         }


    }
    std::cout << "Edit distance is " << distance << std::endl;
    this->distance = distance;
    return ret;

}
```

PS5a: Ring Buffer

Assignment:

The objective of this assignment was to create a simple class named RingBuffer which implements a small API for use in the following assignment, PS5b: Guitar Hero. The Ring Buffer class has six essential member functions, as well as a constructor which takes a single parameter for capacity. The constructor initializes the private data member for capacity to the parameter it is given, then initializes the size to be 0. If the user attempts to initialize a Ring Buffer with a capacity less than 0, an exception will be thrown because it should not be possible to have a negative capacity. The next function, size, is a basic accessor function which returns the object's size. The next two functions isEmpty and isFull check whether or not the Ring Buffer is empty or full, and return the appropriate boolean values. The following function, enqueue, adds an integer to the end of the buffer. Enqueue throws an exception if the buffer is full. The next member function, dequeue, returns the item at the front of the buffer after removing it. The last member function is peek which returns the value at the front of the buffer, but does not actually delete the item. Next, the assignment required that I install cpplint and run the cpplint.py file on my source code to make my code very readable. The last part of the assignment was to use the Boost unit testing framework to test for basic functionality along with whether or not exceptions were being thrown at the proper times.

Key Data Structures, Algorithms and OO Design Patterns:

Vectors were particularly useful for this assignment. Vectors make storing an arbitrary sequence of numbers very easy, however many other data structures such as arrays or lists could also be used for this assignment. Data encapsulation and object oriented design were key topics during this assignment when designing the Ring Buffer. Packaging all necessary parts of the Ring Buffer into a clear, concise API was very useful during the next assignment.

What I Learned:

During this assignment I learned more about the Boost unit testing framework. I had no idea that Boost allows for users to implement test cases which check whether or not an exception is thrown. This was very useful in testing the Ring Buffer class, and ensured the Ring Buffer worked when later used in PS5b: Guitar Hero. Cpplint was also a new experience for me. I have never been required to make my code in a clear, strictly formatted way as cpplint requires. However, I will agree that cpplint's style checking did make my code much more readable.

Evidence of Running Code:

```
Terminal
File  Edit  View  Search  Terminal  Help
lfaletra@dan417-05:~/Downloads/LucasFaletra_PS5a/ps5a$ make
g++ -O3 -std=c++11 -c RingBuffer.hpp RingBuffer.cpp
g++ -O3 -std=c++11 -c RingBuffer.hpp test.cpp
g++ -O3 -std=c++11 -o ps5a RingBuffer.o test.o -lboost_unit_test_framework
lfaletra@dan417-05:~/Downloads/LucasFaletra_PS5a/ps5a$ ./ps5a
Running 9 test cases...

*** No errors detected
lfaletra@dan417-05:~/Downloads/LucasFaletra_PS5a/ps5a$
```

```
C++ = g++ -O3 -std=c++11
BOOST = -lboost_unit_test_framework

all: ps5a

ps5a: RingBuffer.o test.o
        $(C++) -o ps5a RingBuffer.o test.o $(BOOST)

RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
        $(C++) -c RingBuffer.hpp RingBuffer.cpp

test.o: RingBuffer.hpp test.cpp
        $(C++) -c RingBuffer.hpp test.cpp

clean:
        rm ps5a *o *~ *.gch
```

```
// Copyright 2019 Lucas Faletra
#include <stdint.h>
#include <vector>

class RingBuffer {
public:
  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> buffer;
  int size_;
  int capacity;
};
```

```
/*************************
Copyright 2019 Lucas Faletra
************************/
#include "RingBuffer.hpp"
#include <stdint.h>
#include <vector>
#include <stdexcept>

RingBuffer::RingBuffer(int capacity) {
    if (capacity <= 0) {
      throw std::invalid_argument("RB constructor: capacity must be > 0\n");
    }
    this->capacity = capacity;
    this->size_ = 0;
}

int RingBuffer::size() {
  return this->size_;
}

bool RingBuffer::isEmpty() {
  return (this->size_ == 0) ? true : false;
}

bool RingBuffer::isFull() {
  return (this->size_ == this->capacity) ? true : false;
}

void RingBuffer::enqueue(int16_t x) {
  if (this->size_ == this->capacity)
    throw std::runtime_error("enqueue: can’t enqueue to a full ring\n");
  this->buffer.push_back(x);
  this->size_++;
}

int16_t RingBuffer::dequeue() {
  if (this->isEmpty())
    throw std::runtime_error("dequeue: can’t dequeue to an empty buffer");
  int16_t front = this->buffer.at(0);
  this->buffer.erase(this->buffer.begin());
  this->size_--;
  return front;
}

int16_t RingBuffer::peek() {
  if (this->isEmpty())
    throw std::runtime_error("peek: buffer is empty");
  return this->buffer.front();
}
```

```
// Copyright 2019 Lucas Faletra
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <stdint.h>
#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include "RingBuffer.hpp"
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(RingBuffercontructor) {
  BOOST_REQUIRE_NO_THROW(RingBuffer RB(100));

  BOOST_REQUIRE_THROW(RingBuffer RB2(0), std::exception);

  BOOST_REQUIRE_THROW(RingBuffer RB3(0), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
        RingBuffer rb(100);

        rb.enqueue(2);
        rb.enqueue(1);
        rb.enqueue(0);

        BOOST_REQUIRE(rb.dequeue() == 2);
        BOOST_REQUIRE(rb.dequeue() == 1);
        BOOST_REQUIRE(rb.dequeue() == 0);

        BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
}



BOOST_AUTO_TEST_CASE(Test_Constructor_Exceptions) {
  BOOST_REQUIRE_THROW(RingBuffer RB1(-1), std::invalid_argument);
  BOOST_REQUIRE_THROW(RingBuffer RB2(0), std::invalid_argument);

  BOOST_REQUIRE_NO_THROW(RingBuffer RB3(1000));
}

BOOST_AUTO_TEST_CASE(Test_enqueue_exceptions) {
  // putting 4 items into a buffer with capacity 3 should fail
  RingBuffer RB(3);
  RB.enqueue(2);
  RB.enqueue(4);
  RB.enqueue(6);
  BOOST_REQUIRE_THROW(RB.enqueue(8), std::runtime_error);

  // putting 10 items into a buffer with capacity 10 should work
  RingBuffer RB2(10);
  for (int i = 0; i < 10; i++)
    BOOST_REQUIRE_NO_THROW(RB2.enqueue(i * 20));
}

BOOST_AUTO_TEST_CASE(Test_dequeue_exceptions) {
  RingBuffer RB(10);
  BOOST_REQUIRE_THROW(RB.dequeue(), std::runtime_error);
  RB.enqueue(12);
  BOOST_REQUIRE_NO_THROW(RB.dequeue());
```

```
}

BOOST_AUTO_TEST_CASE(Test_peek_exceptions) {
  RingBuffer RB(100);
  BOOST_REQUIRE_THROW(RB.peek(), std::runtime_error);

  RB.enqueue(16);
  BOOST_REQUIRE_NO_THROW(RB.peek());
}

BOOST_AUTO_TEST_CASE(Test_size_accessor) {
  RingBuffer RB(20);
  BOOST_REQUIRE(RB.size() == 0);

  // push an integer i into the buffer. size should be 1, 2, 3 etc.
  for (int i = 0; i < 20; i++) {
    RB.enqueue(i);
    BOOST_REQUIRE(RB.size() == i + 1);
  }
}

BOOST_AUTO_TEST_CASE(Test_isEmpty) {
  RingBuffer RB(10);
  BOOST_REQUIRE(RB.isEmpty() == true);
  RB.enqueue(1);
  BOOST_REQUIRE(RB.isEmpty() == false);
}

BOOST_AUTO_TEST_CASE(Test_isFull) {
  RingBuffer RB(10);
  BOOST_REQUIRE(RB.isFull() == false);
  for (int i = 0; i < 10; i++) {
    RB.enqueue(i);
  }
  BOOST_REQUIRE(RB.isFull() == true);
}
```

PS5b: Guitar Hero

Assignment:

This assignment, Guitar Hero, was to build a fully functional simulation of a Guitar which plays 37 notes of a chromatic scale. Each note corresponds to a specific key on the keyboard. When a key is pressed, the correct note is played. This is accomplished through the use of the Karplus-Strong algorithm. The steps of this algorithm are to fill a buffer with random values to simulate white noise, then perform an averaging operation on the buffer, which functions as a low pass filter for frequencies. After the averaging operation is performed on the buffer, the buffer goes through one more function where each number is scaled by an energy decay value. This produces a sound nearly identical to a note of a guitar. The energy decay factor can also be changed so that the sounds resemble different instruments such as a drum or piano.

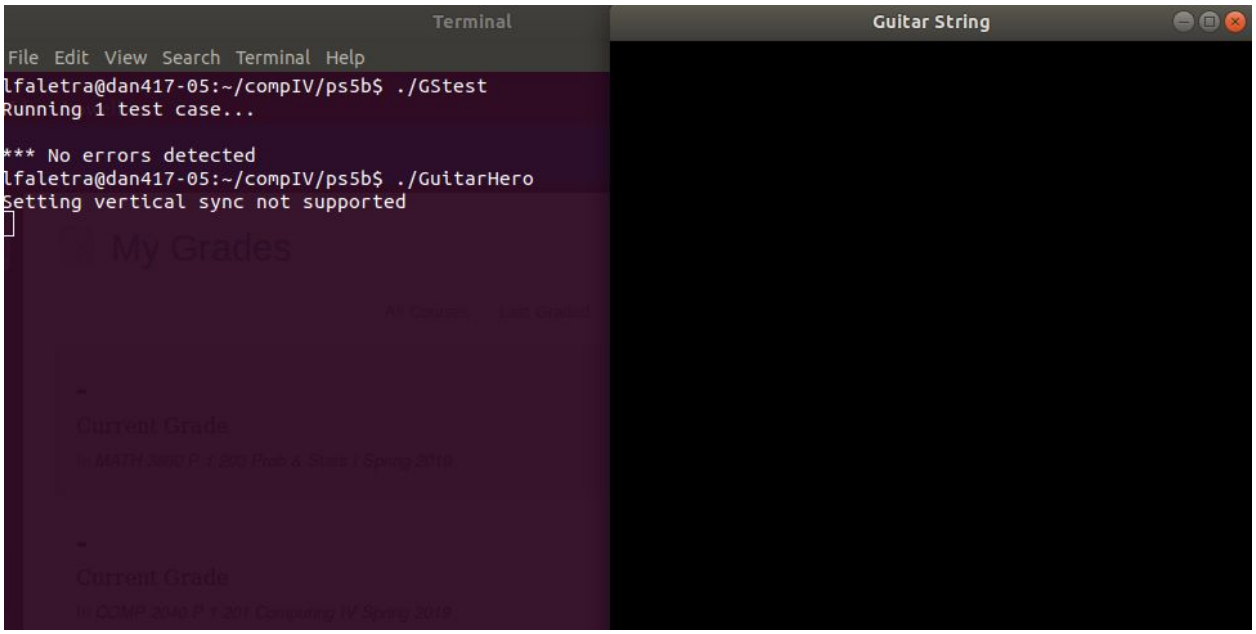
Key Data Structures, Algorithms and OO Design Patterns:

For this assignment, the member functions for std::vector were extremely useful in assessing the Ring Buffer and performing the Karplus-Strong algorithm on the stored data. The use of classes and object oriented design also made this project fairly simple to imagine conceptually, as well as build efficiently.

What I Learned:

During this assignment, I learned how to properly create the Karplus-Strong algorithm to simulate different instruments, using the Ring Buffer class created in the previous assignment. I also learned how to properly use the SFML audio library. The SFML audio Sound Buffer and Sound classes were extremely useful for this assignment.

Evidence of Running Code:



(Due to the nature of this program and the fact that it is meant to only play sound, I can only give a screenshot of the tests passing and the window for Guitar String opening)

```
C++ = g++ -std=c++11
SFML = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system

all: GuitarHero

GuitarHero: GuitarString.o GuitarHero.o RingBuffer.o RingBuffer.o
        $(C++) -o GuitarHero GuitarString.o RingBuffer.o GuitarHero.o  $(SFML)

GuitarHero.o: GuitarHero.cpp GuitarString.hpp RingBuffer.hpp GuitarString.cpp RingBuffer.cpp
        $(C++) -c GuitarHero.cpp GuitarString.hpp RingBuffer.hpp GuitarString.cpp RingBuffer.cpp


GuitarString.o: GuitarString.cpp GuitarString.hpp RingBuffer.hpp RingBuffer.cpp
        $(C++) -c GuitarString.cpp RingBuffer.cpp GuitarString.hpp RingBuffer.hpp

clean:
        rm GuitarHero *.o *~ *.gch
```

```
/****************************
Copyright Lucas Faletra 2019
**************************/
#include "GuitarString.hpp"
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <string>
#include <vector>
#include <list>
#include <iostream>
#include <cmath>
#include <cstddef>


int main() {
  float frequency;
  int i;
  ptrdiff_t index;
  sf::Sound sound;
  sf::SoundBuffer buffer;
  GuitarString G(100);
  std::vector<std::vector<sf::Int16>> samples;
  std::string keys = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
  int arr[] = {16, 28, 22, 4, 30, 17, 31, 19, 24, 33, 20, 34, 8, 35,
                14, 15, 56, 46, 55, 25, 23, 3, 2, 5, 21, 6, 1, 13, 9, 12, 10, 49, 50, 48, 52, 51, 57};

  std::vector<int> numbers;
  for(i = 0; i < 37; i++)
    numbers.push_back(arr[i]);

  // create 37 sample vectors using make samples function
  for(i = 0; i < 37; i++){
    frequency = 440 * (pow(2.0, ((24 - i)/12.0)));
    G = GuitarString(frequency);
    samples.push_back(G.make_samples());
  }

  sf::RenderWindow window(sf::VideoMode(500, 500), "Guitar String");
  while (window.isOpen())
    {
     sf::Event event;
      while (window.pollEvent(event))
      {
        if (event.type == sf::Event::Closed)
          window.close();

        // take an arbitrary key, search the vector of values
        // if the key is not found an error message is displayed
        // otherwise, take the index of the key in the vector of values,
        // then find the corresponding letter

        if (event.type == sf::Event::KeyPressed)
          {
            index = std::find(numbers.begin(), numbers.end(), event.key.code) - numbers.begin();
            if(index < 37)
              {
                buffer.loadFromSamples(&(samples[int(index)][0]), samples[int(index)].size(), 2, 44100);
                sound.setBuffer(buffer);
```

```
            }
             else
                index = -1;

             if(int(index) != -1)
               sound.play();
           }

       }

     }

  return 0;

}
```

```
/***********************
Copyright Lucas Faletra 2019
**********************/
#include "RingBuffer.hpp"
#include <SFML/Audio.hpp>
#include <stdint.h>
#include <vector>
#include <string>

class GuitarString {
public:
  GuitarString(double frequency);
  GuitarString(std::vector<sf::Int16> init);
  void pluck();
  void tic();
  sf::Int16 sample();
  int time_();
  sf::Sound get_sound();
  std::vector<sf::Int16> make_samples();
private:
  std::vector<sf::Int16> samples;
  sf::SoundBuffer buffer;
  sf::Sound sounds;

  RingBuffer* RB;
  int n_tics;
};
```

```
#include <stdint.h>
#include <vector>

class RingBuffer {
public:
  RingBuffer(int capacity);
  int size();
  int cap();
  bool isEmpty();
  bool isFull();
  void enqueue(int16_t x);
  int16_t dequeue();
  int16_t peek();
private:
  std::vector<int16_t> buffer;
  int size_;
  int capacity;
};
```

```
/****************************
Copyright Lucas Faletra 2019
**************************/
//#include "RingBuffer.hpp"
#include "GuitarString.hpp"
#include <vector>
#include <cmath>
#include <SFML/Audio.hpp>
#include <SFML/System.hpp>
#include <stdint.h>
#include <string>
#include <ctime>
#include <cstdlib>
#include <iostream>


GuitarString::GuitarString(double frequency) {
  this->RB = new RingBuffer(ceil(frequency));
}

GuitarString::GuitarString(std::vector<sf::Int16> init) {
  this->samples = init;
  this->RB = new RingBuffer(init.size());
  int i = 0;
  for(i = 0; i < init.size(); i++)
    this->RB->enqueue(int16_t(init[i]));
  //take last sample vector, load buffer
  this->buffer.loadFromSamples(&(this->samples[0]), this->samples.size(), 2,
44100);

  this->sounds.setBuffer(this->buffer);
}

void GuitarString::pluck() {
  //fill rb with random values

  //check if RB is full already
  //if full, empty the whole thing

  if(RB->isFull())
    {
      while(!(RB->isEmpty()))
        {
          RB->dequeue();
        }
    }

  int i = 0;
  // n is random number being pushed into RingBuffer
  int16_t n;

  // seed rand with time for truly random number
  srand(time(0));

  for(i = RB->size(); i < RB->cap(); i++)
    {
      n = ((rand() % 64000) - 32000);
      //n = i;
      this->RB->enqueue(n);
    }

```

```cpp

}

void GuitarString::tic() {
  //do the average function
  int16_t a, b, c;
  //take first and second elements and perform karplus-strong operation
  a = this->RB->dequeue();
  b = this->RB->peek();
  c = int16_t(0.996 * ((a + b) * 0.50));//decay was 0.996 with 0.5
  this->RB->enqueue(c);
  this->n_tics++;
}

sf::Int16 GuitarString::sample() {
  return sf::Int16(this->RB->peek());
}

int GuitarString::time_() {
  return this->n_tics;
}

sf::Sound GuitarString::get_sound(){
  return this->sounds;
}

std::vector<sf::Int16> GuitarString::make_samples() {
  std::vector<sf::Int16> temp;
  this->pluck();
  int i = 0;
  for(i = 0; i < 44100 * 8; i++)
    {
      this->tic();
      this->samples.push_back(this->sample());
    }

  temp = this->samples;
  return temp;
}


```

```
#include "RingBuffer.hpp"
#include <vector>
#include <stdint.h>
#include <stdexcept>

RingBuffer::RingBuffer(int capacity) {

    if(capacity <= 0) {
      throw std::invalid_argument("RingBuffer constructor: capacity must be greater than 0\n");
    }
    this->capacity = capacity;

    this->size_ = 0;

    //make an exception for negative numbers

}

int RingBuffer::size() {
  return this->size_;
}

int RingBuffer::cap() {
  return this->capacity;
}

bool RingBuffer::isEmpty() {
  return (this->size_ == 0) ? true : false;
}

bool RingBuffer::isFull() {
  return (this->size_ == this->capacity) ? true : false;
}

void RingBuffer::enqueue(int16_t x) {
  if(this->size_ == this->capacity)
    throw std::runtime_error("enqueue: can’t enqueue to a full ring\n");
  this->buffer.push_back(x);
  this->size_++;
}

int16_t RingBuffer::dequeue() {
  if(this->isEmpty())
    throw std::runtime_error("dequeue: can’t dequeue to an empty buffer");
  int16_t front = this->buffer.at(0);
  this->buffer.erase(this->buffer.begin());
  this->size_--;
  return front;
}

int16_t RingBuffer::peek() {
  if(this->isEmpty())

    throw std::runtime_error("peek: can’t peek to an empty buffer");
  return this->buffer.front();
}
```

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

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
#include <SFML/Audio.hpp>
#include <SFML/System.hpp>
#include <iostream>
#include <vector>
#include <exception>
#include <stdexcept>
#include <stdint.h>
#include "GuitarString.hpp"

using namespace std;
BOOST_AUTO_TEST_CASE(GS) {
  vector<sf::Int16> v;
  v.push_back(0);
  v.push_back(2000);
  v.push_back(4000);
  v.push_back(-10000);
  BOOST_REQUIRE_NO_THROW(GuitarString gs = GuitarString(v));

  GuitarString gs = GuitarString(v);
  // GS is 0 2000 4000 -10000
  BOOST_REQUIRE(gs.sample() == 0);

  gs.tic();
  // it’s now 2000 4000 -10000 996
  BOOST_REQUIRE(gs.sample() == 2000);

  gs.tic();
  // it’s now 4000 -10000 996 2988
  BOOST_REQUIRE(gs.sample() == 4000);

  gs.tic();
  // it’s now -10000 996 2988 -2988
  BOOST_REQUIRE(gs.sample() == -10000);

  gs.tic();
  // it’s now 996 2988 -2988 -4483
  BOOST_REQUIRE(gs.sample() == 996);

  gs.tic();
  // it’s now 2988 -2988 -4483 1984
  BOOST_REQUIRE(gs.sample() == 2988);

  gs.tic();
  // it’s now -2988 -4483 1984 0
  BOOST_REQUIRE(gs.sample() == -2988);

  // a few more times
  gs.tic();
  BOOST_REQUIRE(gs.sample() == -4483);
  gs.tic();
  BOOST_REQUIRE(gs.sample() == 1984);
```

```
  gs.tic();
  BOOST_REQUIRE(gs.sample() == 0);
}
```

PS6: Airport

Assignment:

The goal of this assignment was to create a running simulation of Logan Airport using low-level programming concepts such as mutex locks and threads. Most of the necessary code for this assignment was provided in the Airplane, AirportServer, and Airport files. The first part of the assignment was to add a lambda expression to the Airport.cpp file. The next part of the assignment was to modify the AirportServer.hpp and AirportServer.cpp files so that mutex variables could be used to successfully lock and unlock specified runways. When a plane requests to land on a specific runway, the request is processed by the program, which then locks mutexes corresponding to that runway and intersecting runways. Multiple threads are used to ensure planes can land simultaneously as well. In total, 7 mutexes are used. Six of these mutexes are for the runways, and the final mutex is used for a condition variable.

Key Data Structures, Algorithms and OO Design Patterns:

Essential data structures for this assignment included mutex locks, threads, and condition variables. Concurrency was a very large portion of this assignment. Concurrency is when a program is able to have multiple processes or copies of a program run simultaneously while also communicating here. This is a very intuitive approach to solving this problem because one of the only alternate solutions would be to have if statements for each case when a plane lands. However, even with a massive amount of if-statements, the program would not be able to have each plane land as a separate process. This makes concurrency the most effective solution to use for the airport and all of its runways.

What I Learned:

This assignment was very interesting because I was able to learn about new topics such as mutex variables, the different kinds of locks such as guard locks, as well as threads and multithreading. This was a nice introduction to these low-level concepts. I also encountered errors with deadlock when creating the simulation. As a result, I learned what deadlock is, what causes deadlock, and how to solve issues arising from deadlock.

Evidence of Running Code:

```
Terminal
File  Edit  View  Search  Terminal  Help
lfaletra@dan417-05:~/compIV/Airport$ make
g++ -c -g -Og -std=c++14 -o Airplane.o Airplane.cpp
g++ -c -g -Og -std=c++14 -o Airport.o Airport.cpp
g++ -c -g -Og -std=c++14 -o AirportRunways.o AirportRunways.cpp
g++ -c -g -Og -std=c++14 -o AirportServer.o AirportServer.cpp
g++ Airplane.o Airport.o AirportRunways.o AirportServer.o -o Airport -pthread
lfaletra@dan417-05:~/compIV/Airport$ ./Airport
Airplane #1 is acquiring any needed runway(s) for landing on Runway 9

Checking airport status for requested Runway 9...
Number of simultaneous landing requests == 1, max == 1
Number of planes landing on runway 4L == 0
Number of planes landing on runway 4R == 0
Number of planes landing on runway 9 == 1
Number of planes landing on runway 14 == 0
Number of planes landing on runway 15L == 0
Number of planes landing on runway 15R == 0
Status check complete, no rule violations (yay!)
Airplane #1 is taxiing on Runway 9 for 10 milliseconds
Airplane #3 is acquiring any needed runway(s) for landing on Runway 9
Airplane #1 is releasing any needed runway(s) after landing on Runway 9
Airplane #1 is waiting for 10 milliseconds before landing again

Checking airport status for requested Runway 9...
```

```
CC = g++
CFLAGS = -c -g -Og -std=c++14
OBJ = Airplane.o Airport.o AirportRunways.o AirportServer.o
DEPS =
LIBS = -pthread
EXE = Airport

all: $(OBJ)
        $(CC) $(OBJ) -o $(EXE) $(LIBS)

%.o: %.cpp $(DEPS)
        $(CC) $(CFLAGS) -o $@ $<

clean:
        rm -f $(OBJ) $(EXE)
```

```
 1: /**
 2: *  Airport driver program
 3: */
 4:
 5: #include <iostream>
 6: #include <thread>
 7: #include <vector>
 8:
 9: #include "AirportServer.h"
10: #include "AirportRunways.h"
11: #include "Airplane.h"
12:
13: using namespace std;
14:
15:
16: void run(Airplane* ap)
17: {
18:         ap->land();
19:
20: } // end run
21:
22:
23: int main(void)
24: {
25:         AirportServer as;
26:
27:         vector<thread> apths; // Airplane threads
28:         auto lambda = [&](auto a) {return a;};
29:                                                     // Create and launch the i
ndividual 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([&](Airplane* x){x->land();}, ap));
35:                 apths.push_back(thread(lambda(run), ap));
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
*  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

```

```
/**
*  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 simulation
        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);

        /**
        *  Check the status of the aiport with respect to any violation of the rules.
        */
        static void checkAirportStatus(RunwayNumber requestedRunway);

        /**
        *  requestRunway() and finishedWithRunway() are helper methods for keeping track of the airport status
        */

        static void requestRunway(RunwayNumber rn)
        {
                runwayInUse[rn]++;

        } // end useRunway()


        static void finishedWithRunway(RunwayNumber rn)
        {
                runwayInUse[rn]--;

        } // end finishedWithRunway()


        static int getNumLandingRequests()
        {
```

```
                return numLandingRequests;
        }


        static void incNumLandingRequests()
        {
                numLandingRequests++;
                if (numLandingRequests > maxNumLandingRequests)
                        maxNumLandingRequests = numLandingRequests;
        }


        static void decNumLandingRequests()
        {
                numLandingRequests--;
        }

private:

        /**
         *  The following variables and methods are used to detect violation
s of the rules of this simulation.
         */

        static int runwayInUse[NUM_RUNWAYS]; // Keeps track of how many airp
lanes are attempting to land on a given runway

        static int numLandingRequests; // Keeps track of the number of simul
taneous landing requests

        static int maxNumLandingRequests; // Keeps track of the max number o
f simultaneous landing requests

}; // end class AirportRunways

#endif

```

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

#ifndef AIRPORT_SERVER_H
#define AIRPORT_SERVER_H

#include <mutex>
#include <random>
#include <condition_variable>

#include "AirportRunways.h"

class AirportServer
{
public:

        /**
        *  Default constructor for AirportServer class
        */
        AirportServer()
        {

                // ***** Initialize any Locks and/or Condition Variables here as necessary *****

        } // end AirportServer default constructor


          /**
          *  Called by an Airplane when it wishes to land on a runway
          */
        void reserveRunway(int airplaneNum, AirportRunways::RunwayNumber runway);

        /**
        *  Called by an Airplane when it is finished landing
        */
        void releaseRunway(int airplaneNum, AirportRunways::RunwayNumber runway);


private:

        // Constants and Random number generator for use in Thread sleep calls
        static const int MAX_TAXI_TIME = 10; // Maximum time the airplane will occupy the requested runway after landing, in milliseconds
        static const int MAX_WAIT_TIME = 100; // Maximum time between landings, in milliseconds

        /**
        *  Declarations of mutexes and condition variables
        */
        mutex runwaysMutex; // Used to enforce mutual exclusion for acquiring & releasing runways

        //condition variable
        condition_variable cv0;
        //one mutex corresponding to each runway
```

```
   55:         mutex mx1, mx2, mx3, mx4, mx5, mx6, mx7;
   56:         /**
   57:          *  ***** Add declarations of your own Locks and Condition Variables
 here *****
   58:          */
   59:
   60: }; // end class AirportServer
   61:
   62: #endif
```

```
#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(AirportRunways::RUNWAY_4L, AirportRunways::RUNWAY_15R);

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

                apServ->reserveRunway(airplaneNum, requestedRunway);

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

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

        } // end while

} // end Airplane::land
```

```
 1: #include "AirportRunways.h"
 2:
 3: int AirportRunways::runwayInUse[AirportRunways::NUM_RUNWAYS];
 4:
 5: int AirportRunways::numLandingRequests = 0;
 6:
 7: int AirportRunways::maxNumLandingRequests = 0;
 8:
 9: mutex AirportRunways::checkMutex;
10:
11:
12: string 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 rul
es.
37:   */
38: void 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 " << runwayN
ame(requestedRunway) << "..." << endl;
45:
46:         requestRunway(requestedRunway);
47:
48:         // Check the number of landing requests
49:         cout << "Number of simultaneous landing requests == " << numLandingR
equests
50:                  << ", max == " << maxNumLandingRequests << endl;
51:
52:         if (numLandingRequests > MAX_LANDING_REQUESTS)
53:         {
54:                 cout << "***** The number of simultaneous landing requests e
xceeds Air Traffic Control limit of " << MAX_LANDING_REQUESTS << "!\n";
55:                 crash = true;
56:         }
57:
```

```
   58:         // Check the occupancy of each runway
   59:         for (int i = RUNWAY_4L; i <= RUNWAY_15R; i++)
   60:         {
   61:                 cout << "Number of planes landing on runway " << runwayName(
RunwayNumber(i)) << " == " << runwayInUse[i] << endl;
   62:
   63:                 if (runwayInUse[i] > 1)
   64:                 {
   65:                         cout << "***** The number of planes landing on runwa
y " << runwayName(RunwayNumber(i)) << " is greater than 1!\n";
   66:                         crash = true;
   67:                 }
   68:         }
   69:
   70:         // Check individual restrictions on each runway
   71:         if ((runwayInUse[RUNWAY_9] > 0)
   72:                 && ((runwayInUse[RUNWAY_4R] > 0) || (runwayInUse[RUNWAY_15R]
 > 0)))
   73:         {
   74:                 cout << "***** Runways 9, 4R, and/or 15R may not be used sim
ultaneously!\n";
   75:                 crash = true;
   76:         }
   77:
   78:         if (((runwayInUse[RUNWAY_15L] > 0) || (runwayInUse[RUNWAY_15R] > 0))
   79:                 && ((runwayInUse[RUNWAY_4L] > 0) || (runwayInUse[RUNWAY_4R]
> 0)))
   80:         {
   81:                 cout << "***** Runways 15L or 15R may not be used simultaneo
usly with Runways 4L or 4R!\n";
   82:                 crash = true;
   83:         }
   84:
   85:         // If any of the rules have been violated, terminate the simulation
   86:         if (crash)
   87:         {
   88:                 cout << "***** CRASH! One or more rules have been violated.
Due to the crash, the airport is closed!\n";
   89:                 exit(-1); // Abnormal program termination
   90:         }
   91:
   92:         // Status check is normal
   93:         cout << "Status check complete, no rule violations (yay!)\n";
   94:
   95: } // end AirportRunways::checkAirportStatus()
```

```
#include <iostream>
#include <mutex>
#include <thread>
#include <condition_variable>

#include "AirportServer.h"

//uncommented runways.unlock, moved finished with runway after unlock statem
ents

/**
*  Called by an Airplane when it wishes to land on a runway
*/
void AirportServer::reserveRunway(int airplaneNum, AirportRunways::RunwayNum
ber runway)
{
        // Acquire runway(s)
        {  // Begin critical region
                unique_lock<mutex> runwaysLock(runwaysMutex);

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

                        cout << "Airplane #" << airplaneNum << " is acquirin
g any needed runway(s) for landing on Runway "
                                << AirportRunways::runwayName(runway) << en
dl;
                }
                /**
                 *  ***** Add your synchronization here! *****
                 */

                unique_lock<mutex> cond_lock(mx7);
                AirportRunways::incNumLandingRequests();

                while(AirportRunways::getNumLandingRequests() > AirportRunwa
ys::MAX_LANDING_REQUESTS)
                {
                      cv0.wait(cond_lock);
                }
                //AirportRunways::incNumLandingRequests();


                //AirportRunways::checkAirportStatus(runway);


                //AirportRunways::incNumLandingRequests();
                if(runway == 0){
                    mx1.lock();
                    mx5.lock();
                    mx6.lock();
                }
                if(runway == 1){
                    mx2.lock();
                    mx3.lock();
                    mx5.lock();
                    mx6.lock();
                }
                if(runway == 2){
                    mx2.lock();
                    mx3.lock();
```

```
                    mx6.lock();
                }
                if(runway == 3){
                  mx4.lock();
                }
                if(runway == 4){
                    mx1.lock();
                    mx2.lock();
                    mx5.lock();
                }
                if(runway == 5){
                    mx1.lock();
                    mx2.lock();
                    mx3.lock();
                    mx6.lock();
                }

                // Check status of the airport for any rule violations
                //AirportRunways::incNumLandingRequests();
                AirportRunways::checkAirportStatus(runway);
                //AirportRunways::incNumLandingRequests();
                runwaysLock.unlock();
                //cond_lock.unlock();
                //cv0.notify_all();

        } // 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);

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

                cout << "Airplane #" << airplaneNum << " is taxiing on Runway " << AirportRunways::runwayName(runway)
                         << " for " << taxiTime << " milliseconds\n";
        }

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

} // end AirportServer::reserveRunway()


 /**
  *  Called by an Airplane when it is finished landing
  */
void AirportServer::releaseRunway(int airplaneNum, AirportRunways::RunwayNumber runway)
{

        // Release the landing runway and any other needed runways
        { // Begin critical region
          //AirportRunways::finishedWithRunway(runway);
```

```
          //AirportRunways::decNumLandingRequests();

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

                        cout << "Airplane #" << airplaneNum << " is releasin
g any needed runway(s) after landing on Runway "
                                << AirportRunways::runwayName(runway) << en
dl;
                }

                /**
                *  ***** Add your synchronization here! *****
                */
                //AirportRunways::finishedWithRunway(runway);

                if(runway == 0)
                  {
                    mx1.unlock();
                    mx5.unlock();
                    mx6.unlock();
                  }

                if(runway == 1){
                      mx2.unlock();
                      mx3.unlock();
                      mx5.unlock();
                      mx6.unlock();
                }

                if(runway == 2)
                  {
                    mx2.unlock();
                    mx3.unlock();
                    mx6.unlock();
                  }

                if(runway == 3)
                  {
                      mx4.unlock();
                  }

                if(runway == 4)
                  {
                      mx1.unlock();
                      mx2.unlock();
                      mx5.unlock();
                  }

                if(runway == 5)
                  {
                      mx1.unlock();
                      mx2.unlock();
                      mx3.unlock();
                      mx6.unlock();
                  }

                AirportRunways::decNumLandingRequests();
```

```
                AirportRunways::finishedWithRunway(runway);


                //AirportRunways::decNumLandingRequests();
                //runwaysLock.unlock();
                // Update the status of the airport to indicate that the lan
ding is complete

                //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);

        // Wait for a random number of milliseconds before requesting the ne
xt 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()
```

PS7: Kronos Intouch Parsing

Assignment:

The goal of this assignment was to read through a log file from a Kronos Time Clock, parse the log file correctly, and output a text file report chronologically using expression parsing and the Boost library. I used Boost date and time functions to compute elapsed time between the server boot and boot completion. I used two regular expressions to search for matches between an expected start message and expected end message. The program searches for error messages and outputs whether the boot was a success or failure by writing the information to the file.

Key Data Structures, Algorithms and OO Design Patterns:

Regexes were both essential and required for this assignment to parse the information in the log file properly. The two regular expressions I created are used to search for particular start and end strings in the log file. File streams were also particularly useful in both reading the log file and writing to the .rpt file. No class was required for this assignment, so I decided to place all necessary methods in the main.cpp file for ease of access. As a result, no particular object oriented designs were used in my implementation aside from using a vector to store the hours, minutes and seconds, and using numerous strings to hold important information from the log file.

What I Learned:

During this assignment I learned what regexes are and how to use them. Regular expressions were vital to this assignment because of the need to search for specific occurrences of strings in the log file. Regexes from the Boost library made this process very simple. Without regular expressions, numerous strings would have to be created for every possible sequence of numbers or letters in the format of information being searched for in the log file.

Evidence of Running Code:

(On the following page you will find the output file "device1_intouch.log.rpt" for the input file device1_intouch.log)

```
Terminal
File  Edit  View  Search  Terminal  Help
lfaletra@dan417-05:~/compIV/ps7$ make
g++     -c -o main.o main.cpp
g++ -std=c++14 -O3 -o PS7 main.o -lboost_regex -lboost_date_time
lfaletra@dan417-05:~/compIV/ps7$ ./PS7 device1_intouch.log
lfaletra@dan417-05:~/compIV/ps7$
```

emacs@dan417-05

File Edit Options Buffers Tools Help

Save Undo

```
Device booted
435368(device1_intouch.log.rpt): 2014-03-25 Boot Start
435758(device1_intouch.log.rpt): 2014-03-25 19:15:02 Boot Completed
        Boot Time: 183000ms

Device booted
436499(device1_intouch.log.rpt): 2014-03-25 Boot Start
436858(device1_intouch.log.rpt): 2014-03-25 19:32:44 Boot Completed
        Boot Time: 165000ms

Device booted
440718(device1_intouch.log.rpt): 2014-03-25 Boot Start
440790(device1_intouch.log.rpt): 2014-03-25 22:04:27 Boot Completed
        Boot Time: 161000ms

Device booted
440865(device1_intouch.log.rpt): 2014-03-26 Boot Start
441215(device1_intouch.log.rpt): 2014-03-26 12:50:29 Boot Completed
        Boot Time: 167000ms

Device booted
442093(device1_intouch.log.rpt): 2014-03-26 Boot Start
442431(device1_intouch.log.rpt): 2014-03-26 20:44:13 Boot Completed
        Boot Time: 159000ms

Device booted
443072(device1_intouch.log.rpt): 2014-03-27 Boot Start
443410(device1_intouch.log.rpt): 2014-03-27 14:11:42 Boot Completed
        Boot Time: 161000ms
```

-:--- **device1_intouch.log.rpt** All L1 (Fundamental)

```
C++ = g++ -std=c++14 -O3
BOOST = -lboost_regex -lboost_date_time

all: PS7

PS7: main.o
        $(C++) -o PS7 main.o $(BOOST)

stdin_boost.o:  main.cpp
        $(C++) -c main.cpp

clean:
        rm *~ *.o PS7
```

```
 1: // Copyright Lucas Faletra 2019
 2: #include <iostream>
 3: #include <fstream>
 4: #include <string>
 5: #include <vector>
 6: #include <boost/regex.hpp>
 7: #include "boost/date_time/gregorian/gregorian.hpp"
 8: #include "boost/date_time/posix_time/posix_time.hpp"
 9:
10:
11:
12: using boost::posix_time::time_duration;
13:
14: int main(int argc, char* argv[]) {
15:   // input output files
16:   std::ifstream log(argv[1]);
17:   std::string out_name = std::string(argv[1]);
18:   out_name += ".rpt";
19:   std::ofstream out(out_name.c_str());
20:   // counters
21:   int lines_read = 0;
22:   int boot_successes = 0;
23:   int boot_total = 0;
24:   // empty strings used later
25:   std::string rep, line, boots, start_date, end_date, full_date;
26:   boost::smatch sm;
27:   // vector to hold hours minutes seconds
28:   std::vector<int> hms;
29:   hms.resize(3);
30:   // variables for time taken to complete and dates
31:   boost::posix_time::ptime begin, end;
32:   boost::gregorian::date d1, d2;
33:   boost::posix_time::time_duration time_difference;
34:   bool found_start = false;
35:   // first string in file probably something like this
36:   std::string start_string = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
37:   start_string += "([0-9]{2}):([0-9]{2}):([0-9]{2}): \\(log.c.166\\) ";
38:   start_string += "server started";
39:   // make a regex for the start string
40:   boost::regex start_regex(start_string, boost::regex::perl);
41:   // end string should be something like this
42:   std::string end_string = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
43:   end_string += "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}):INFO:oejs.";
44:   end_string += "AbstractConnector:Started SelectChannelConnector@0.0.0.0:90
80";
45:   // make a regex for the last string
46:   boost::regex end_regex(end_string);
47:
48:   // read info while log is open, line by line
49:   if (log.is_open()) {
50:     while (getline(log, line)) {
51:       // clear strings for dates each iteration
52:       start_date.clear();
53:       end_date.clear();
54:
55:       // perform operations on start regex
56:       if (boost::regex_search(line, sm, start_regex)) {
57:         start_date = sm[1] + "-" + sm[2] + "-" + sm[3];
58:         end_date += " " + sm[4] + ":" + sm[5] + ":" + sm[6];
59:
60:         full_date = sm[1] + "-" + sm[2] + "-" + sm[3];
```

```
        d1 = boost::gregorian::date
          (boost::gregorian::from_simple_string(full_date));
      // reset hours minutes seconds in vector
        hms[0] = std::stoi(sm[4]);
        hms[1] = std::stoi(sm[5]);
        hms[2] = std::stoi(sm[6]);
       // set begin time
        begin = boost::posix_time::ptime
           (d1, time_duration(hms[0], hms[1], hms[2]));

       // if error found, append incomplete to boot string
        if (found_start == true) {
          boots += "Incomplete boot\n";
        }
       // append the starting boot info to the boot string
        boots += "Device booted\n";
        boots += std::to_string(lines_read) + "(" + out_name + "): ";
        boots += start_date + " Boot Start\n";

       // increment # of boots
        boot_total++;
        found_start = true;
      }
      // check for a match based on regex
       if (boost::regex_match(line, sm, end_regex)) {
       // if match, append the elements of the smatch variable
        end_date = sm[1] + "-" + sm[2] + "-" + sm[3];
        end_date += " " + sm[4] + ":" + sm[5] + ":" + sm[6];
        // append date elements to full_date
        full_date = sm[1] + "-" + sm[2] + "-" + sm[3];
       // set the second date variable
        d2 = boost::gregorian::date
           (boost::gregorian::from_simple_string(full_date));

      // store time back into the vector in hours, minutes, seconds
        hms[0] = std::stoi(sm[4]);
        hms[1] = std::stoi(sm[5]);
        hms[2] = std::stoi(sm[6]);

        end = boost::posix_time::ptime
          (d2, time_duration(hms[0], hms[1], hms[2]));

        boots += std::to_string(lines_read) + "(" + out_name + "): ";
        boots += end_date + " Boot Completed\n";

        time_difference = end - begin;

        // append boot time
        boots += "\tBoot Time: ";
        boots += std::to_string
          (time_difference.total_milliseconds()) + "ms \n\n";

        // boot successful, increment counter
        boot_successes++;
        // reset bool value to false for successful boot
        found_start = false;
       }

       // increment lines read from log
      lines_read++;
    }
```

```
    // close the log
    log.close();
  }
  boots.erase(boots.end()-1);
  rep += boots;
  // write the information to the output log.rpt file
  out << rep;
  // close it and finish
  out.close();
  return 0;
}
```