Maxwell Rider

Computing IV portfolio

Spring 2019

Contents:

PS0: SFML Introduction

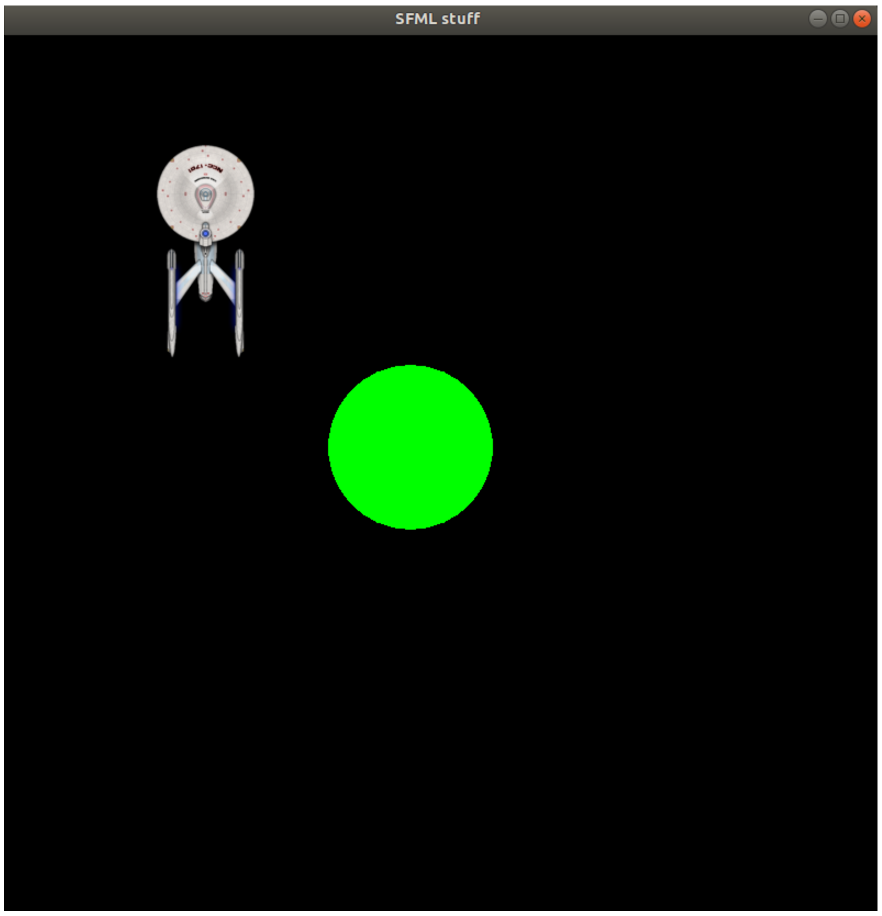
Assignment Overview:

This assignment was created to help with understanding how to use some of the functions in SFML. The assignment was to draw a circle using SFML to make sure we had it installed correctly. We had to extend the demo and add some additional functionality such as drawing an image sprite, making it move, and make it respond to key strokes.

In my version of the assignment I had a circle that would change colors depending on what key was pressed (‘r’ for red, ‘g’ for green, or ‘b’ for blue). I also had a sprite image of the Starship Enterprise that would respond to arrow key presses, moving in the direction pressed.

What I learned:

- How to draw shapes in SFML
- How to load and draw images in SFML
- How to use key events in SFML
- How to create a window in SFML



```
CC = g++
CFLAGS = -c -g -Og -Wall -Werror -ansi -pedantic
OBJ = main.o
DEPS =
LIBS = -lsfml-graphics -lsfml-window -lsfml-system
EXE = SFML-app

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

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

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

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

int main()
{
    sf::RenderWindow window(sf::VideoMode(800, 800), "SFML stuff");
    sf::CircleShape circle;

    circle.setRadius(75);
    circle.setPosition(300,300);
    circle.setFillColor(sf::Color::Green);


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



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


        window.clear();


        //move sprite
        if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
        {
          //move sprite up
          sprite.move(0,-1);
        }
        if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
        {
          //move sprite down
          sprite.move(0,1);
        }
        if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
        {
          //move sprite left
          sprite.move(-1,0);
        }
        if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
        {
          //move sprite right
          sprite.move(1,0);
        }
        if(sf::Keyboard::isKeyPressed(sf::Keyboard::A))
        {
          //rotate sprite left
          sprite.rotate(-0.5);
        }
        if(sf::Keyboard::isKeyPressed(sf::Keyboard::D))
        {
          //rotate sprite right
          sprite.rotate(0.5);
        }

        //Change cirlce color
        if(sf::Keyboard::isKeyPressed(sf::Keyboard::R))
        {
          //make circle red
          circle.setFillColor(sf::Color::Red);
        }
        if(sf::Keyboard::isKeyPressed(sf::Keyboard::G))
        {
          //make circle red
          circle.setFillColor(sf::Color::Green);
        }
        if(sf::Keyboard::isKeyPressed(sf::Keyboard::B))
        {
          //make circle red
          circle.setFillColor(sf::Color::Blue);
        }


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

    return 0;
}
```

PS1: LFSR Image Encoding

Assignment Overview:

This was a two-part assignment. The first part was to create a Linear Feedback Shift Register (LFSR). The second part was to use the LFSR to assist in encoding an image of our choosing.

Part A: Part A of the assignment was to implement a Linear Feedback Shift Register as well as file built specifically to test it. An LFSR is something that takes a string of bits as an input. To shift the bit string, it moves all the bits one place to the left. It then takes the XOR of the removed bit and a bit at a given tap position and adds it to the end of the string.

My implementation involved using a string object and manipulating that to accomplish my needs. This also involved converting from a string value to an int and vice versa (For the calculations and subsequent string concatenation).

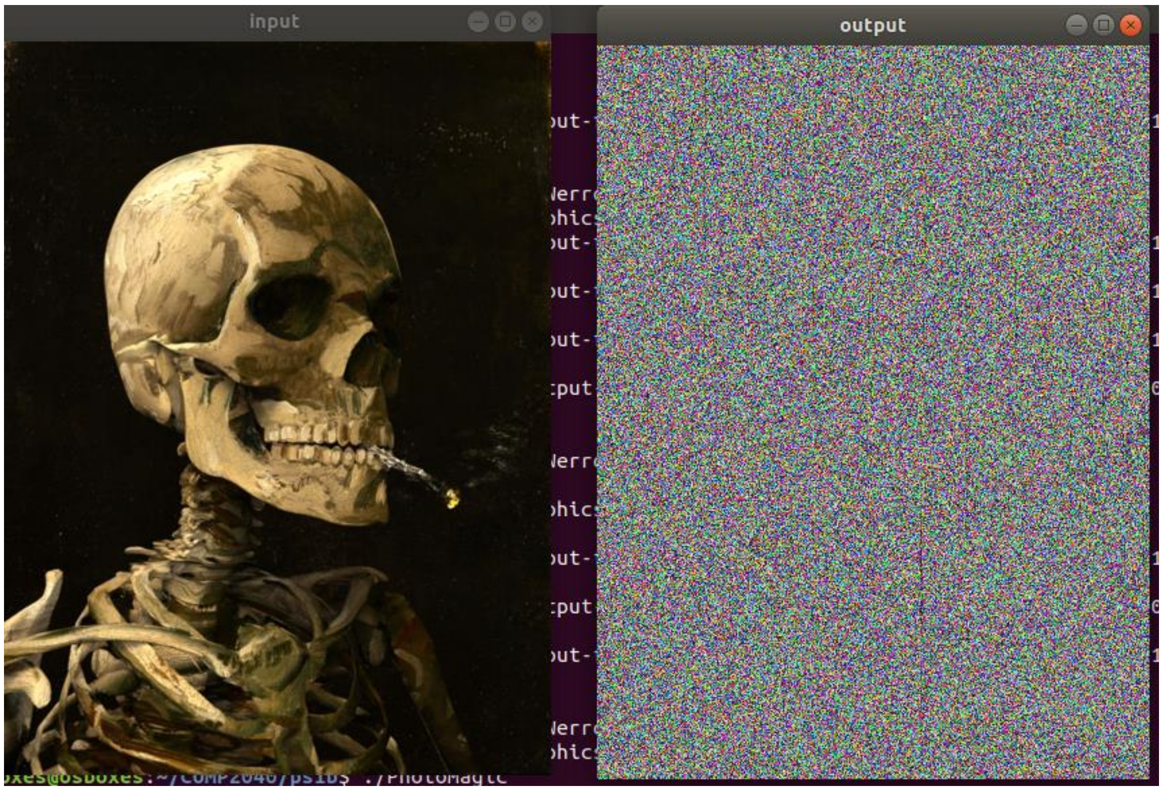
Part B: Part B of the assignment was using the previously created LFSR to encode/decode an image file.

This was simple, I took the color value of each pixel and would XOR it with a newly generated 8-bit number that was created by the LFSR. This results in a pseudo-randomization of the pixel colors that makes the original image unidentifiable.

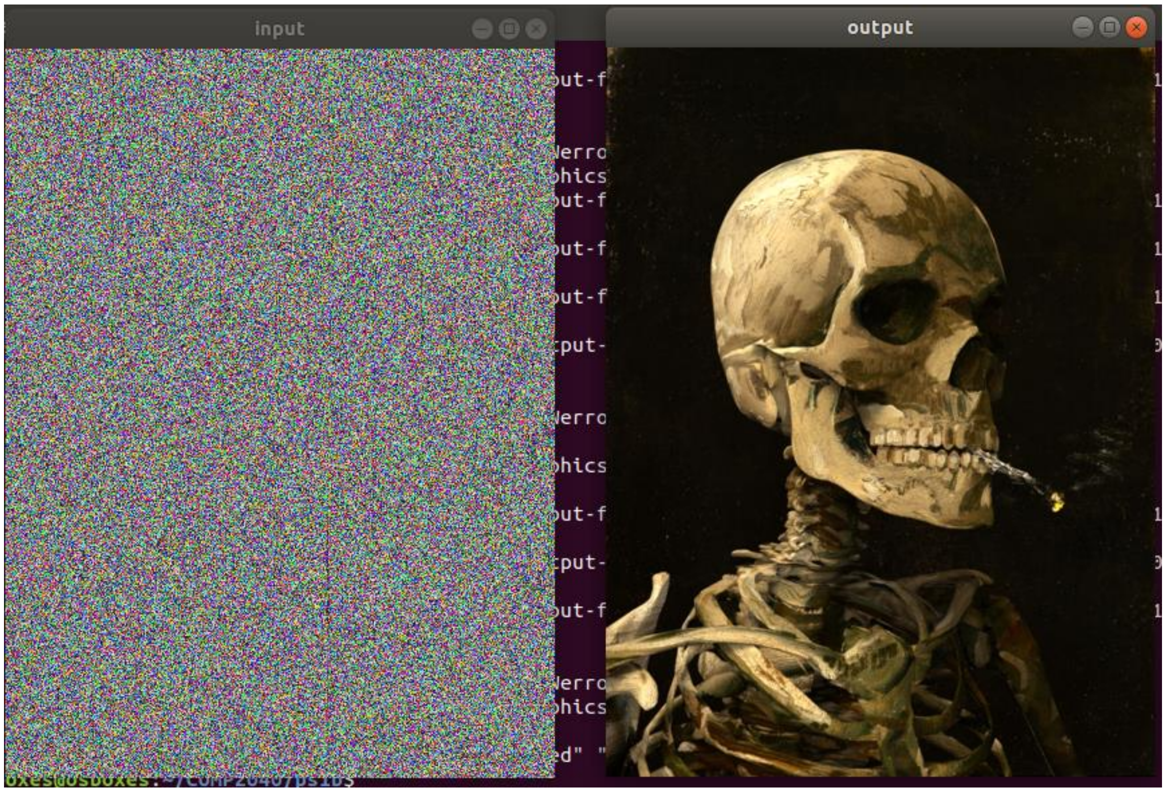
What I Learned:

- Basic string manipulation (Part A)
- How to accept command line arguments (Part B)
- How to use the “getPixel()” and “setPixel()” functions provided by SFML (Part B)

PS1: LFSR Image Encoding



```
oxes@osboxes:~/COMP2040/ps1b$ ./PhotoMagic
```



```
oxes@osboxes:~/COMP2040/ps1b$
```

```
CC = g++
CFLAGS = -c -g -Og -Wall -Werror -ansi -pedantic
SFML = -lsfml-graphics -lsfml-window -lsfml-system
EXE = PhotoMagic

all: PhotoMagic

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

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

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

clean:
	rm *.o
	rm PhotoMagic
```

```
#ifndef LFSR_HPP
#define LFSR_HPP

#include <iostream>

using namespace std;



class LFSR
{
public:
  LFSR(string seed, int t);
  int step();
  int generate(int k);

  //overloaded stream insertion operator
  friend ostream& operator<< (ostream &out, LFSR &lfsr);

private:
  int tapPos;            //position to be tapped
  string bitString;      //holds the bit string
};

#endif
```

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

//Constructor
LFSR::LFSR(string seed, int t)
{
  bitString = seed;
  tapPos = t;
}

int LFSR::step()
{

  char newBitChar;
  int newBitNum;

  //get the index of the bit to tap
  int tapIndex = ((bitString.length() - 1) - tapPos);


  //XOR the 'left-most' bit with the tapped bit
  int newBit = (bitString[0] ^ bitString[tapIndex]);

  //cout << newBit << endl;

  //store newBit value for return
  newBitNum = newBit;

  //convert newBit to char
  newBitChar = newBit + '0';

  //'shift' the bits to the left by one
  bitString.erase(bitString.begin()+0);

  //replace the right most bit with the result of the XOR
  bitString.push_back(newBitChar);


  return newBitNum;
}

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

  for(int i = 0; i < k; i++)
    {
        x = (x * 2) + step();
    }
  return x;
}

ostream& operator<< (ostream &out, LFSR &lfsr)
{
  out << lfsr.bitString;
  return out;
}
```

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

int main(int argc, char* argv[])
{
  //SFML variables
  sf::Image inputImage;
  sf::Image outputImage;
  sf::Color p;

  //make sure the correct number of arguments is read
  if(argc != 5)
    {
      cout << "Usage: "<< argv[0] <<" [inputFile] [outputFile] [seed] Tap Po
sition] \n";
      return -1;
    }

  //save arguments from commandline into variables
  string inputFile = argv[1];
  string outputFile = argv[2];
  string seed = argv[3];
  int tapPos = atoi(argv[4]);

  LFSR encoder(seed, tapPos); //call LFSR with the seed and tap position

  if (!inputImage.loadFromFile(inputFile)) //Make sure the file is read
    return -1;

  if (!outputImage.loadFromFile(inputFile))
    return -1;

  sf::Vector2u size = inputImage.getSize();
  sf::RenderWindow  window1(sf::VideoMode(size.x, size.y), "input");
  sf::RenderWindow  window2(sf::VideoMode(size.x, size.y), "output");

  //"randomize" the pixel colors
  for(int x = 0; x < (signed)size.x; x++)
    {
      for (int y = 0; y < (signed)size.y; y++)
        {
          p = inputImage.getPixel(x,y);
          p.r ^= encoder.generate(tapPos);
          p.g ^= encoder.generate(tapPos);
          p.b ^= encoder.generate(tapPos);
          outputImage.setPixel(x,y,p);
        }
    }

  sf::Texture inputTexture;
  sf::Texture outputTexture;
  inputTexture.loadFromImage(inputImage);
  outputTexture.loadFromImage(outputImage);

  sf::Sprite inputSprite;
  sf::Sprite outputSprite;
  inputSprite.setTexture(inputTexture);
  outputSprite.setTexture(outputTexture);

  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(inputSprite);
      window1.display();
      window2.clear();
      window2.draw(outputSprite);
      window2.display();
    }

  if(!outputImage.saveToFile(outputFile))
     return -1;

  return 0;
}
```

PS2: Pythagoras Tree

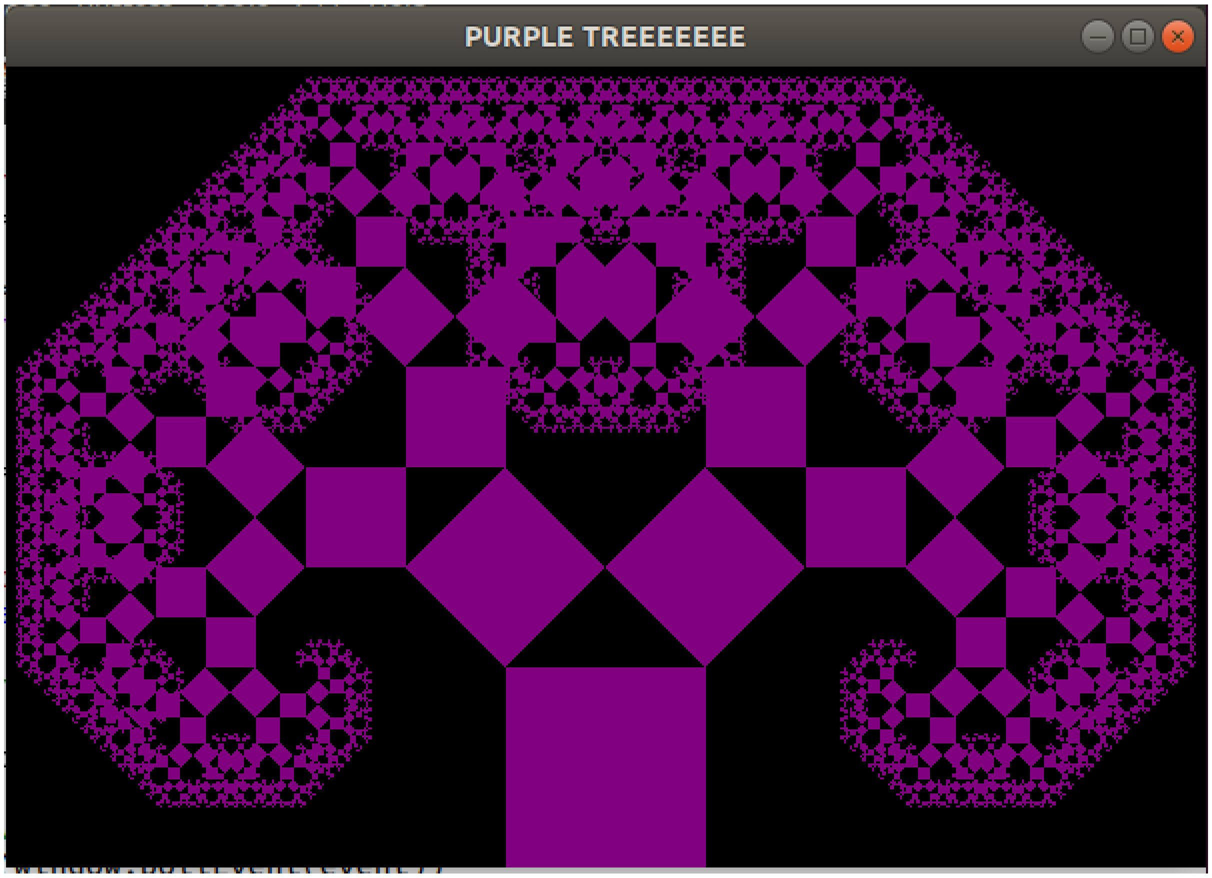
Assignment Overview:

This assignment was to use recursion to create a Pythagoras Tree, also known as a fractal tree. The program would accept an initial size of the box and depth as command line arguments and then create a seemingly infinite number of boxes branching off of each other getting progressively smaller and smaller.

In my implementation I would recursively create the boxes at the correct points, doing some fun math to make sure the points for each box were correct. I would then push each freshly created box onto a vector to store it to be printed out (drawn) later. I also implemented a draw function that used a for loop to iterate through the vector of boxes and draw each one as it was called.

What I Learned:

- More practice with command line arguments
- How to use the Vector2F coordinate function provided by SFML
- General practice with geometry



```
CC = g++
CFLAGS  = -Wall -Werror -ansi -pedantic
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system

all: ptree

ptree: PTree.o
	$(CC) PTree.o -o tree $(LFLAGS)

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

clean:
	rm *.o
	rm ptree
```

```
#ifndef PTREE_HPP
#define PTREE_HPP

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

using namespace std;

class ptree : public sf::Drawable
{
public:
  ptree(int size, int depth);
  void pTree(sf::Vector2f p1, sf::Vector2f p2, int depth);
  void draw(sf::RenderTarget& target, sf::RenderStates states) const;

private:
  int _depth;
  vector <sf::ConvexShape> _boxVec; //vector to hold box objects
  sf::ConvexShape _box;
  sf::Vector2f _p1;
  sf::Vector2f _p2;
};

#endif
```

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

using namespace std;

ptree::ptree(int size, int depth)
{
  _p1 = sf::Vector2f(size * 2.5, size * 4);
  _p2 = sf::Vector2f(size * 3.5, size * 4);

  this->_depth = depth;
  _box.setPointCount(4);
  pTree(_p1, _p2, _depth);
}


//recursive function
void ptree::pTree(sf::Vector2f p1, sf::Vector2f p2, int depth)
{
  //base case for recursion
  if(depth > 0)
    {
      //define the position of the next points
      sf::Vector2f p3 = sf::Vector2f(p2.x - (p1.y - p2.y), p2.y - (p2.x - p1
.x));
      sf::Vector2f p4 = sf::Vector2f(p1.x - (p1.y - p2.y), p1.y - (p2.x - p1
.x));
      sf::Vector2f p5 = sf::Vector2f(p4.x + (p2.x - p1.x - (p1.y -p2.y))/2,
p4.y - (p2.x - p1.x + p1.y - p2.y)/2);

      //set the points of the box to be drawn
      _box.setPoint(0,p1);
      _box.setPoint(1,p2);
      _box.setPoint(2,p3);
      _box.setPoint(3,p4);
      _box.setFillColor(sf::Color(128,0,128)); //set the color to purple

      //store the _box object on a vector
      _boxVec.push_back(_box);

      //recursively call the ptree function for left then right
      pTree(p4, p5, depth - 1);
      pTree(p5, p3, depth - 1);
    }
}


void ptree::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
  //use a for loop to access and print out _box objects
  for(int i = 0; i < (signed)_boxVec.size(); i++)
    {
      target.draw(_boxVec.at(i));
    }
}

int main(int argc, char* argv[])
{
  //make sure there are three arguments taken in
  if(argc != 3)
    {
      cout << argv[0] << " [side length of first box] [recursion depth] \n";
      return -1;
    }

  //convert the arguments to ints
  int size = atoi(argv[1]);
  int depth = atoi(argv[2]);

  //create ptree object
  ptree tree(size, depth);

  sf::RenderWindow window(sf::VideoMode(size * 6,size * 4), "PURPLE TREEEEEE
E");

  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;
}
```

PS3: N-Body simulation

Assignment Overview:

This was another two-part assignment. Part A consisted of simply loading images into a window with their sizes and positions all given by a text file. Part B involved adding the physics simulation. Ultimately we were supposed to create a semi accurate model of the first four planets in the solar system revolving around the sun.

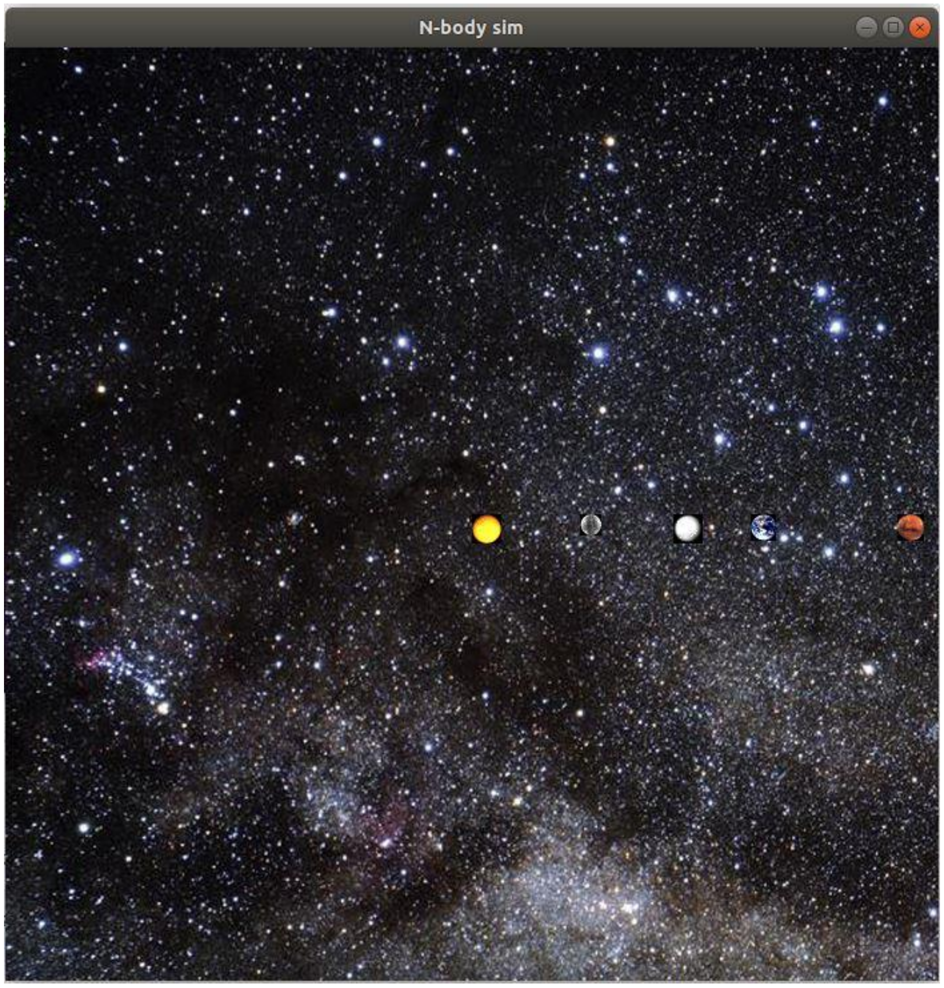
Part A: This part of the assignment was to load in a static universe of an N-Body simulation. All the information needed to do so was given in a text document that had the (X,Y) position of each planet, it's mass, its X/Y velocity, and finally the name of the file we had to load in. To read in this information the stream insertion operator (>>) was overloaded to accept input from a text file. I had a body class that would create bodies with the given values. In a loop, I would create a pointer to a body object, and then load all the required information into it using the >> operator. I would then set the correct starting position of the planet (Based off the text file) and then push the body object onto a vector. I then used my main function to draw each planet into the universe.

Part B: The second part of the assignment involved adding the physics to the simulation in order to get the planets moving. This part also allowed the program to accept a total time for the simulation as well as a time step amount. Based on the information we had we needed to calculate the new position of the planets, force, the new velocity, and acceleration. I was unable to get my assignment fully working due to a problem with the way that I updated my position of the planets.

What I Learned:

- How to overload an operator to read in specific information from a file
- More practice with drawing images in SFML

PS3: N-Body simulation



There were two parts but because I couldn't get the second part working the pictures are the same, I chose not to include it because it seemed redundant to do so.

```
CC = g++
CFLAGS = -Wall -Werror -ansi -pedantic
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio

all: NBody

NBody: body.o
	$(CC) body.o -o NBody $(LFLAGS)

body.o: body.cpp body.hpp
	$(CC) -c body.cpp $(CLFAGS)

clean:
	rm *.o
	rm NBody
```

```cpp
#ifndef BODY_HPP
#define BODY_HPP

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

using namespace std;

const int windowLength = 700;
const int windowHeight = 700;

//force of gravity on an object
const double Fg = 6.67e-11;

class body : public sf::Drawable
{
public:

  body();

  body(double xPos, double yPos, double xVel,
       double yVel, double mass, string fileName);

  void setPlanetPos(double universeRadius);

  //moves the objects based on given velocities and calculated forces
  void step(double seconds);
  void calcVel(double xAccel, double yAccel, double deltaT);

  //calculate the gravitational force on an object
  friend double calcXForce(body &b1, body &b2);
  friend double calcYForce(body &b1, body &b2);
  void setForce(double xForce, double yForce);

  friend istream &operator>> (istream &input, body &cBody);

  //for deugging purposes
  friend ostream &operator<< (ostream &output, body &cBody);

private:
  void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const
;

  double _planetNum;
  double _uniRadius;
  double _xPos;
  double _yPos;
  double _xVel;
  double _yVel;
  double _mass;
  string _fileName;

  double _xForce;
  double _yForce;
  double _xAccel;
  double _yAccel;

  sf::Image _image;
  sf::Sprite _sprite;
  sf::Texture _texture;
  };
#endif
```

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

body::body(){}

body::body(double xPos, double yPos, double xVel,
           double yVel, double mass, string fileName)
{
  _xPos = xPos;
  _yPos = yPos;
  _xVel = xVel;
  _yVel = yVel;
  _mass = mass;
  _fileName = fileName;

  if(!_image.loadFromFile(fileName))
    {
      return;
    }

  _texture.loadFromImage(_image);
  _sprite.setTexture(_texture);

  _sprite.setPosition(sf::Vector2f(_xPos, _yPos));
}

void body::setPlanetPos(double universeRadius)
{

  // cout << "setPos xPos: " << _xPos << endl;
  // cout << "setPos yPos: " << _yPos << endl;


  //calculate x and y postion of new objects
  _xPos = ((_xPos/universeRadius) * (windowLength/2)) + (windowLength/2);
  _yPos = ((_yPos/universeRadius) * (windowHeight/2)) + (windowHeight/2);

  // cout << "_xPos/universeRadius: " << (_xPos/universeRadius) << endl;
  // cout << "setPos xPos: " << _xPos << endl;
  // cout << "setPos yPos: " << _yPos << endl;
  // cout << "Radius: " << universeRadius << endl;

  // sf::FloatRect spriteDim = _sprite.getGlobalBounds();
  // _xPos -= (spriteDim.width/2);
  // _yPos -= (spriteDim.height/2);

  //set postion
  _sprite.setPosition(sf::Vector2f(_xPos, _yPos));
}

//calculate the gravitational force on an object
double calcXForce(body &b1, body &b2)
{
  double dx, dy; //deltaX and delta Y
  double r2, r; //radius squared and radius
  double F; //force
  double Fx; //force of x

  //calculate delta x and delta y
  dx = fabs(b2._xPos - b1._xPos);
  dy = fabs(b2._yPos - b1._yPos);

  //calculate radius squared
  r2 = (pow(dx,2) + pow(dy,2));

  //calculate radius between two objects
  r = sqrt(r);

  //calculate the force between two objects
  F = (Fg * b1._mass * b2._mass)/r2;

  //calculate the force on the x-axis
  Fx = F * (dx/r);

  return Fx;
}

double calcYForce(body &b1, body &b2)
{
  double dx, dy; //deltaX and delta Y
  double r2, r; //radius squared and radius
  double F; //force
  double Fy; //force of y

  //calculate delta x and delta y
  dx = fabs(b2._xPos - b1._xPos);
  dy = fabs(b2._yPos - b1._yPos);

  //calculate radius squared
  r2 = (pow(dy,2) + pow(dy,2));

  //calculate radius between two objects
  r = sqrt(r);

  //calculate the force between two objects
  F = (Fg * b1._mass * b2._mass)/r2;

  //calculate the force on the y-axis
  Fy = F * (dy/r);

  return Fy;
}

void body::setForce(double xForce, double yForce)
{
  _xForce = xForce;
  _yForce = yForce;
}

void body::step(double seconds)
{
  //calculate acceleration
  _xAccel = _xForce / _mass;
  _yAccel = _yForce / _mass;

  //calculate velocity
  calcVel(_xAccel, _yAccel, seconds);

  //calculate the new position
  _xPos += (_xVel * seconds);
  _yPos += (_yVel * seconds);

}

void body::calcVel(double xAccel, double yAccel, double deltaT)
{
  _xVel += (deltaT * xAccel);
  _yVel += (deltaT * yAccel);
}

void body::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
  target.draw(_sprite);
}

istream &operator>>(istream &input, body &pBody)
{
  input >> pBody._xPos >> pBody._yPos; //read in x and y positions
  input >> pBody._xVel >> pBody._yVel; //read in x and y velocity
  input >> pBody._mass; //read in mass of planets/objects
  input >> pBody._fileName; //read in .gif file name

  //create texture from .gif file
  if(!pBody._image.loadFromFile(pBody._fileName))
    {
      return input;
    }

  pBody._texture.loadFromImage(pBody._image);
  pBody._sprite.setTexture(pBody._texture);
  pBody._sprite.setPosition(sf::Vector2f(pBody._xPos, pBody._yPos));

  pBody._xForce = 0;
  pBody._yForce = 0;
  pBody._xAccel = 0;
  pBody._yAccel = 0;

  return input;
}

//used to debug the program
ostream& operator<< (ostream &output, body &cBody)
{
  output << "_xPos|_yPos|_xVel|_yVel|_mass|_fileName\n";
  output << cBody._xPos << "|" << cBody._yPos << "|" << cBody._xVel << "|" <
< cBody._yVel
         << "|" << cBody._mass << "|" << cBody._fileName << endl;

  return output;
}


int main(int argc, char* argv[])
{
  if(argc != 3)
    {
      cout << "Usage: ./Nbody [Total sim time] [Time step] < planets.txt\n";
      return -1;
    }

  string totalSimTime(argv[1]);
  string stepTime(argv[2]);
  string::size_type sz;

  double elapsedSimTime  = 0;
  double simTime = stod(totalSimTime, &sz);
  double timeStep = stod(stepTime, &sz);

  string planetCount, radius;

  cin >> planetCount;
  cin >> radius;

  int planetNum = stoi(planetCount, &sz);
  double universeRadius = stod(radius, &sz);

  vector<body> bodyVec;

  for(int i = 0; i < planetNum; i++)
    {
      body* p = new body();

      cin >> *p;

      p->setPlanetPos(universeRadius);
      bodyVec.push_back(*p);
    }

  sf::RenderWindow window(sf::VideoMode(windowLength, windowHeight), "N-Body
sim");

  sf::Image backgroundImage;

  if(!backgroundImage.loadFromFile("space.jpg"))
    {
      return -1;
    }

  sf::Texture backgroundTexture;
  backgroundTexture.loadFromImage(backgroundImage);

  sf::Sprite backgroundSprite;
  backgroundSprite.setTexture(backgroundTexture);

  sf::Font timeFont;
  timeFont.loadFromFile("arial.ttf");

  sf::Text timeText;
  timeText.setFont(timeFont);
  timeText.setCharacterSize(14);
  timeText.setColor(sf::Color::White);

  // sf::Music music;
  // if(!music.openFromFile("2001.ogg"))
  //   {
  //     return -1;
  //   }
  // music.play();

  vector<body>::iterator iter,x,y;
```

```

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

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


       window.clear();

       window.draw(backgroundSprite);

       timeText.setString("Elapsed Time: " + to_string(elapsedSimTime));
       window.draw(timeText);

       x = bodyVec.begin();
       double xForce, yForce;


       if(elapsedSimTime <= simTime)
         {

       for(int i = 0; i < planetNum; i++)
         {
           y = bodyVec.begin();
           xForce = 0;
           yForce = 0;

           for(int j = 0; j < planetNum; j++)
             {
               if(i != j)
                 {
                   xForce += calcXForce(*x, *y);
                   yForce += calcYForce(*x, *y);
                 }
               y++;
             }
           x->setForce(xForce, yForce);
           x++;
           xForce = 0;
           yForce = 0;
         }

       // for(iter = bodyVec.begin(); iter != bodyVec.end(); iter++)
       //         {
       //           iter -> step(timeStep);
       //           iter -> setPlanetPos(universeRadius);
       //         }

       for(iter = bodyVec.begin(); iter != bodyVec.end(); iter++)
         {
           window.draw(*iter);
           // iter -> step(timeStep);
           // iter -> setPlanetPos(universeRadius);
```

```
            }


        //I know this is not eh correct way to do this but it’s the only way I could get the planets to display at all
        iter -> step(timeStep);
        iter -> setPlanetPos(universeRadius);

        window.display();

        elapsedSimTime += timeStep;
          }
      }

    for(iter = bodyVec.begin(); iter != bodyVec.end(); iter++)
      {
        cout << *iter << endl;
      }
    return 0;
  }
```

PS4: DNA Sequence Alignment

Assignment Overview:

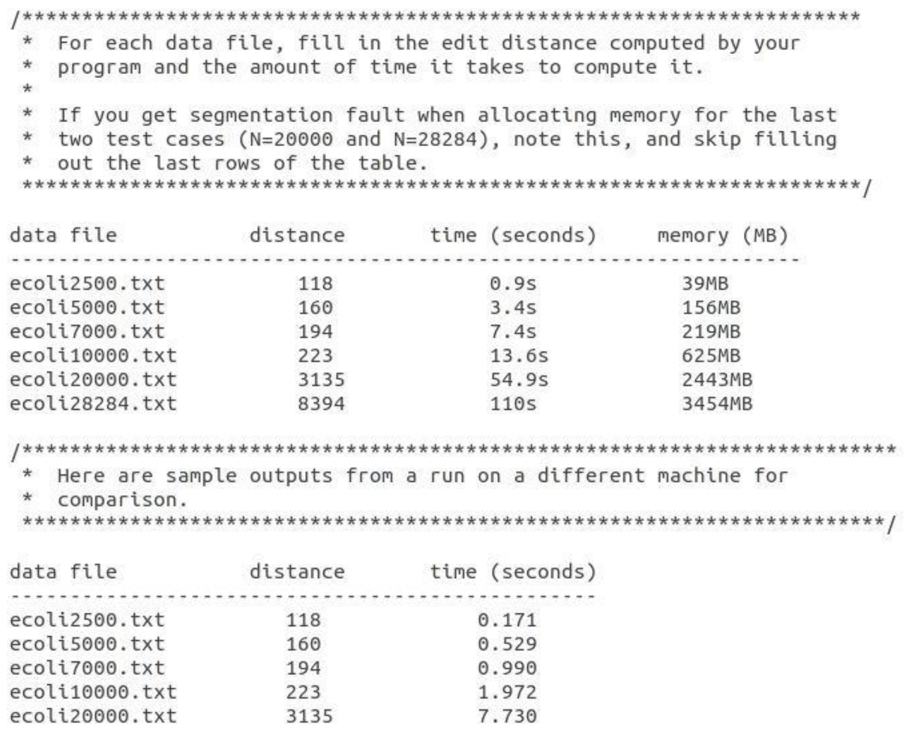
This assignment was to write a program that would find the best alignment of two DNA sequences using the Edit Distance method to calculate their similarity. We are allowed to place gaps in the sequence to give them an equal length, this also serves to help give a more similar alignment. Given a text file that contained two strings of letters corresponding to gene types, we compare the two strings similarity and report the version that has the most similar alignment as well as its Edit Distance.

In my implementation I used the formula for finding the edit distance. I create a 2D vector to store all the values for my edit distance calculation. I calculate what each value my three options would give me and then take the minimum of those three. The value calculated by each option is partly calculated using the penalty amount. I calculated the penalty for each alignment (0 for the alignment being correct, 1 for incorrect). I do these calculations starting at the last block in the 2D Vector and work my way up to the [0][0] value.

What I learned:

- How to use the edit distance formula
- Practice reading information from files
- Practice using 2D Vectors

PS4: DNA Sequence Alignment



```
/**********************************************************************
 *  For each data file, fill in the edit distance computed by your
 *  program and the amount of time it takes to compute it.
 *
 *  If you get segmentation fault when allocating memory for the last
 *  two test cases (N=20000 and N=28284), note this, and skip filling
 *  out the last rows of the table.
 **********************************************************************/

data file           distance       time (seconds)     memory (MB)
-------------------------------------------------------------------
ecoli2500.txt          118             0.9s            39MB
ecoli5000.txt          160             3.4s            156MB
ecoli7000.txt          194             7.4s            219MB
ecoli10000.txt         223             13.6s           625MB
ecoli20000.txt         3135            54.9s           2443MB
ecoli28284.txt         8394            110s            3454MB

/*************************************************************************
 *  Here are sample outputs from a run on a different machine for
 *  comparison.
 *************************************************************************/

data file           distance       time (seconds)
--------------------------------------------------
ecoli2500.txt         118             0.171
ecoli5000.txt         160             0.529
ecoli7000.txt         194             0.990
ecoli10000.txt        223             1.972
ecoli20000.txt        3135            7.730
```

```
CC = g++
CFLAGS = -Wall -Werror -ansi -pedantic -g
LFLAGS = -lsfml-system

all: ED

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

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

clean:
	rm *.o
	rm ED
```

```
#ifndef ED_HPP
#define ED_HPP

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

using namespace std;

class ED
{
public:
  ED();//default constructor
  ED(string stringA, string stringB);//value constructor
  static int penalty(char a, char b);//returns penalty for aligning chars a and b (either 1, 0)
  static int min(int x, int y, int z);//returns the minimum of the three arguments
  int optDistance();//populates matrix and returns the optimal distance
  string alignment();//trace matrix and return a string of the best alignment
  void print();

private:
  string _stringA, _stringB;
  vector<vector<int>> _matrix;
};

#endif
```

```cpp
#include "ED.hpp"

ED::ED(){}

ED::ED(string stringA, string stringB)
{
  _stringA = stringA;
  _stringB = stringB;
}

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

  if(a == b)
    {
      penalty = 0;
    }
  else if(a != b)
    {
      penalty = 1;
    }

  return penalty;
}

int ED::min(int x, int y, int z)
{
  if(x < y && x < z)
    {
      return x;
    }
  else if(y < x && y < z)
    {
      return y;
    }
  else if(z < x && z < y)
    {
      return z;
    }

  //in the case that all numbers are equal return the first value
  return x;
}

int ED::optDistance()
{
  int M = _stringB.length();
  int N = _stringA.length();

  for(int i = 0; i <= M; i++)
    {
      vector<int> placeHolder;
      _matrix.push_back(placeHolder);

      for(int j = 0; j <= N; j++)
        {
          _matrix.at(i).push_back(0);
        }
    }

  for(int i = 0; i<= M; i++)
    {
      _matrix[i][N] = 2 * (M - i);
    }

  for(int j = 0; j <= N; j++)
    {
      _matrix[M][j] = 2 * (N - j);
    }

  for(int i = M - 1; i >= 0; i--)
    {
      for(int j = N - 1; j >= 0; j--)
        {
          int opt1 = _matrix[i+1][j+1] + penalty(_stringA[j] , _stringB[i]);
          int opt2 = _matrix[i][j+1] + 2;
          int opt3 = _matrix[i+1][j] + 2;

          _matrix[i][j] = min(opt1, opt2, opt3);
        }
    }

  return _matrix[0][0];
}

string ED::alignment()
{
  ostringstream retString;
  int M = _stringB.length();
  int N = _stringA.length();
  int matchPenalty;
  string finalAlignment;

  int opt1, opt2, opt3;
  int i = 0, j = 0;

  while(i < M || j < N)
    {
      try
        {
          matchPenalty = penalty(_stringA[j], _stringB[i]);
          opt1 = _matrix.at(i+1).at(j+1) + matchPenalty;
        }
      catch(const out_of_range& error)
        {
          opt1 = -1;
        }
      try
        {
          opt2 = _matrix.at(i+1).at(j) + 2;
        }
      catch(const out_of_range& error)
        {
          opt2 = -1;
        }
      try
        {
          opt3 = _matrix.at(i).at(j+1) + 2;
        }
      catch(const out_of_range& error)
        {
```

```cpp
          opt3 = -1;
        }

      if (_matrix[i][j] == opt1)
        {
          retString << _stringA[j] << " " << _stringB[i] << " " << matchPenalty << "\n";
          i++;
          j++;
        }
      else if(_matrix[i][j] == opt2)
        {
          retString << "- " << _stringB[i] << " 2\n";
          i++;
        }
      else if(_matrix[i][j] == opt3)
        {
          retString << _stringA[j] << " -" << " 2\n";
          j++;
        }
    }
  finalAlignment = retString.str();
  return finalAlignment;
}

void ED::print()
{
  vector<vector<int>>::iterator A;
  vector<int>::iterator B;

  for(A = _matrix.begin(); A != _matrix.end(); A++)
    {
      for(B = (*A).begin(); B != (*A).end(); B++)
        {
          cout << *B << " ";
        }
      cout << "\n";
    }
}

int main(int agrc, char* argv[])
{
  sf::Clock clock;
  sf::Time time;

  string stringA, stringB, alignment;

  int distance;

  cin >> stringA >> stringB;

  ED sequencePair(stringA, stringB);

  distance = sequencePair.optDistance();

  alignment = sequencePair.alignment();

  cout << "Edit Distance: " << distance << endl;
  cout << alignment;
  cout << "Edit distance: " << distance << endl;

  //sequencePair.print();

  time = clock.getElapsedTime();
  cout << "\nExecution Time: " << time.asSeconds() << " seconds\n";


  return 0;
}
```

PS5: Guitar Hero

Assignment Overview:

This was another two-part assignment. Part A was simply creating a basic ringbuffer using any method we wanted. Part B was using that ringbuffer to implement a Guitar Hero type system where presses on the keyboard would correspond to a specific note being played.

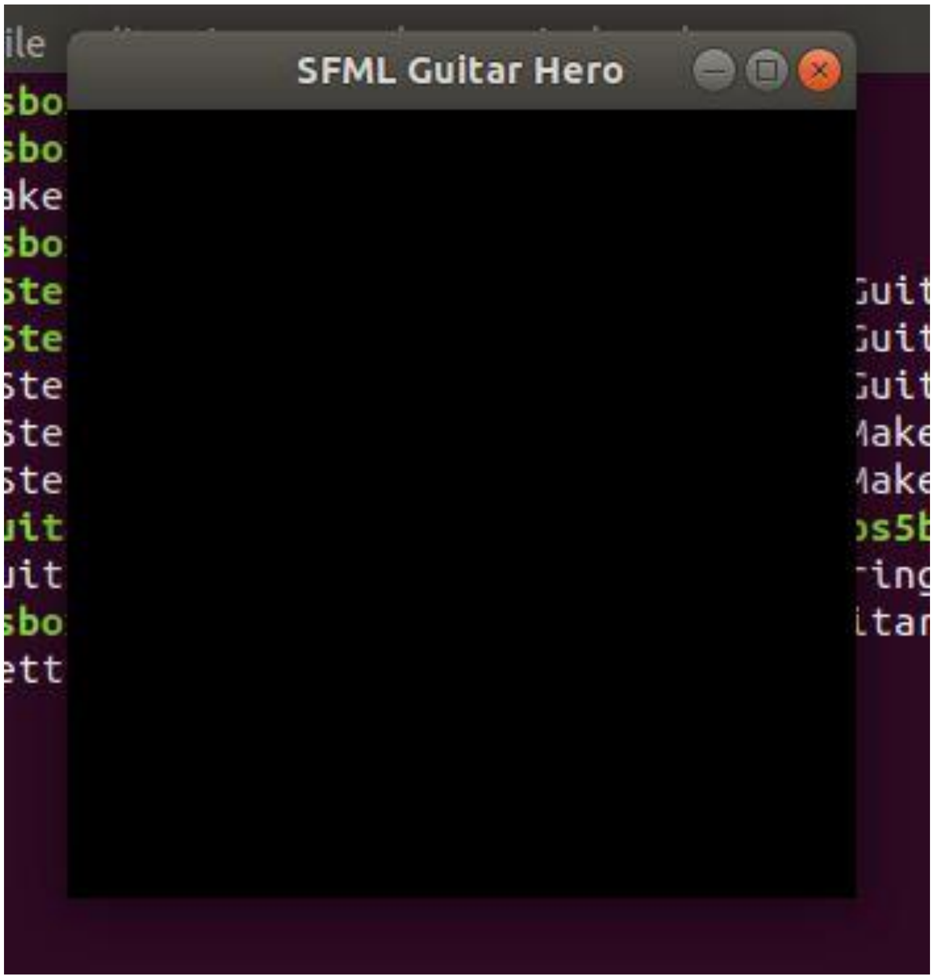
Part A: Part A was very simple, we only had to create a basic ringuffer along with the “helper” functions that went with it. I implemented mine using a vector. We defined the functions enqueue, dequeue, size, isFull, and isEmpty. We also had to make sure that our implementation passed a test file. There wasn’t much to this part of the assignment.

Part B: This was a much bigger assignment than part A. We had to use the ringbuffer we created to simulate a guitar being plucked. In my implementation I first create a guitar string buffer of ringbuffers. I then perform several operations on the ringbuffers including filling it random values to simulate white noise, and pulling values out of it to get samples. All of the operations done are based off of the Karplus-Strong algorithm I then used the SFML TextEntered function to match certain key presses to certain sounds.

What I Learned:

- How to use the Karplus-Strong algorithm
- How to use the SFML TextEntered function
- How to play audio in SFML

PS5: Guitar Hero



This is the only evidence I could think to attach (other than the code itself). I can’t exactly play a song in a word doc)

```
CC = g++
CFLAGS = -Wall -Werror -ansi -pedantic -std=gnu++11
BOOST = -lboost_unit_test_framework
SFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio

all: GuitarHero GStest

GuitarHero: GuitarHero.o GuitarString.o ringBuffer.o
	$(CC) GuitarHero.o GuitarString.o ringBuffer.o -o GuitarHero $(SFLAGS)

GStest: GStest.o GuitarString.o ringBuffer.o
	$(CC) GStest.o GuitarString.o ringBuffer.o -o GStest $(BOOST)


#GUITAR_HERO FILE
GuitarHero.o: GuitarHero.cpp GuitarString.hpp
	$(CC) -c GuitarHero.cpp GuitarString.hpp $(CFLAGS)

#IMPLEMENTATION FILES
GuitarString.o: GuitarString.cpp GuitarString.hpp
	$(CC) -c GuitarString.cpp GuitarString.hpp $(CFLAGS)

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

GStest.o: GStest.cpp
	$(CC) -c GStest.cpp  $(BOOST)

clean:
	rm *.o
	rm GuitarHero
	rm GStest
```

```cpp
#ifndef RINGBUFFER_HPP
#define RINGBUFFER_HPP

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

using namespace std;

class RingBuffer
{
public:
  RingBuffer(int capacity); //Creates empty ring buffer with size of given c
apacity
  int size(); //Return the current size
  bool isEmpty(); //Return true if size  == 0
  bool isFull(); //Return true if size == capacity
  void enqueue(int16_t x); //Add item x to the end
  int16_t dequeue(); //Delete item from front and return it
  int16_t peek(); //Returns the item from the front (but does not delete)
  void empty(); //Sets the _head and _tail to 0 and sets
private:
  int _head;
  int _tail;
  int _size;
  int _capacity;
  vector<int16_t>_buf;
  bool _full;
};

#endif
```

```cpp
/*
Copyright 2019 Maxwell Rider
 */
#include "ringBuffer.hpp"

RingBuffer::RingBuffer(int capacity) {
  if (capacity < 1) {
      throw
        invalid_argument("RB constructor capaciy must be greater than zero")
;
    }

  _capacity = capacity;
  _head = 0;
  _tail = 0;
  _size = 0;
  _buf.resize(capacity);
}

int RingBuffer::size() {
  return _size;
}

bool RingBuffer::isEmpty() {
  if (_size == 0) {
    return true;
  } else {
    return false;
  }
}

bool RingBuffer::isFull() {
  if (_size == _capacity) {
    return true;
  } else {
    return false;
  }
}

void RingBuffer::enqueue(int16_t x) {
  if (isFull()) {
      throw
        runtime_error("Enqueue: can’t enqueue to a full ring");
    }

  if (_tail >= _capacity) {
    _tail = 0;
  }

  _buf.at(_tail) = x;
  _tail++;
  _size++;
}

int16_t RingBuffer::dequeue() {
  if (isEmpty()) {
      throw
        runtime_error("Dequeue: can’t dequeue from an empty ring");
    }
  int16_t num = _buf.at(_head);
  _buf.at(_head) = 0;
  _head++;
  _size--;

  // checks if _head needs to loop back to the begining of the ring
  if (_head >= _capacity) {
      _head = 0;
    }

  return num;
}

int16_t RingBuffer::peek() {
  if (isEmpty()) {
      throw
        runtime_error("Peek: can’t peek from an empty ring");
    }
  return _buf.at(_head);
}

```

```cpp
#ifndef GUITARHERO_HPP
#define GUITARHERO_HPP

#include <iostream>
#include <vector>
#include <cmath>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>
#include "ringBuffer.hpp"

using namespace std;

const int SAMPLING_RATE = 44100;
const double ENERGY_DECAY = 0.996;

class GuitarString
{
public:
  explicit GuitarString(double frequency); //Create guitar string of given frequency using a sampling rate of 44,100
  explicit GuitarString(vector<sf::Int16> init); //Create a guitar string with size and initial values given by the vector
  ~GuitarString();

  void pluck(); //Pluck the guitar string by replacing the buffer with random values
  void tic(); //Advance the simulation by one time step
  sf::Int16 sample(); //Return the current sample
  int time(); //Return the number of times tic was called so far

private:
  RingBuffer* _buffer;
  int _time; //increases when tic() is called
};

#endif
```

```
/*
Copyright 2019 Maxwell Rider
 */


#include "GuitarString.hpp"
#include <random>

GuitarString::GuitarString(double frequency) {
  //Create a buffer with a capacity based off of the the given frequency
  int size = ceil(SAMPLING_RATE/frequency);
  _buffer = new RingBuffer(size);

  //Fill the buffer with zeros casted to type "int16_t"
  for (int i = 0; i < size; i++) {
    (*_buffer).enqueue((int16_t)0);
  }

  //Set time count to 0
  _time = 0;
}

GuitarString::GuitarString(vector<sf::Int16> init) {

  int bufSize = init.size();

  //Create a buffer with a capacity equal to that of the given vector
  _buffer = new RingBuffer(bufSize);

  vector<sf::Int16>::iterator iter;

  //Fill the buffer with values from the given vector
  for (iter = init.begin(); iter != init.end(); iter++) {
    (*_buffer).enqueue((int16_t)*iter);
  }

  //Set time count to 0
  _time = 0;
}

void GuitarString::pluck() {

  random_device rd;
  mt19937 gen(rd());
  uniform_int_distribution<int> dist(-32768, 32767);

  //Store the size of the buffer in a variable
  int bufSize = (*_buffer).size();

  //empty _buffer using the empty() function in "ringBuffer.cpp"
  while(!(*_buffer).isEmpty()) {
    (*_buffer).dequeue();
  }

  //Fill it with random values using "std::uniform_int_distribution<int> dis
t(start, end)
  for (int i = 0; i < bufSize; i++) {
    if (!(*_buffer).isFull()) {
      (*_buffer).enqueue(dist(gen));
    }
  }
}

void GuitarString::tic() {

  //The following is using the Karplus-Strong algorithm

  // int16_t num1;
  // int16_t num2;
  // int16_t avg;
  // int16_t val

  //Retrieve the first two values from the buffer
  if (!(*_buffer).isEmpty()) {
    int16_t num1 = (*_buffer).dequeue();
    int16_t num2 = (*_buffer).peek();

    //average the two values
    int16_t avg = (num1 + num2)/2;

    //Multiply by the energy decay factor (0.996)
    int16_t val = ENERGY_DECAY * avg;

    //Put that value into the buffer
    if (!(*_buffer).isFull()) {
      (*_buffer).enqueue((sf::Int16)val);
    }
    //increase time counter
     _time++;
  }
}

sf::Int16 GuitarString::sample() {
  sf::Int16 sample;
  if (!(*_buffer).isEmpty()) {
    sample = (*_buffer).peek();
  }
  return sample;
 }

int GuitarString::time() {
  //Return the amount of times tic() was called
  return _time;
}

GuitarString::~GuitarString() {
  //delete _buffer;
}
```

```
/*
  GUITAR HERO FILE
  USED TO RUN "GuitarString.cpp"
 */

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

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

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

#include "ringBuffer.hpp"
#include "GuitarString.hpp"

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

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

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

int main(int argc, char * argv[]) {
  sf::RenderWindow window(sf::VideoMode(300,300), "SFML Guitar Hero");
  sf::Event event;
  double freq;

  vector<sf::Int16> sample;
  vector<vector<sf::Int16>> samples(NUM_OF_KEYS);
  vector<sf::SoundBuffer> soundBuf(NUM_OF_KEYS);
  vector<sf::Sound> sounds(NUM_OF_KEYS);

  string keys = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/’ ";

  for (int i = 0; i < (signed)keys.size(); i++) {

    freq = CONCERT_A * pow(2, (i -24)/12);

    GuitarString gs = GuitarString(freq);

    sample = makeSamplesFromString(gs);
    samples[i] = sample;

    if (!soundBuf[i].loadFromSamples(&samples[i][0], samples[i].size(), 1, SAMPLES_PER_SEC)) {
        throw runtime_error("sf::SoundBuffer: failed to load from samples");
    }
    //sf::Sound sound2;
    //sound2.setBuffer(soundBuf[i]);
    //sounds[i] = sound2;
    sounds[i].setBuffer(soundBuf[i]);
  }

  while (window.isOpen()) {
    while (window.pollEvent(event)) {
      switch (event.type) {
      case sf::Event::Closed:
        window.close();
        break;
      case sf::Event::TextEntered: {
        size_t found = keys.find(static_cast<char>(event.text.unicode));
        if (found != string::npos) {
          cout << found << endl;
          sounds[found].play();
        }
        break;
      }
      default:
        break;
      }
    window.clear();
    window.display();
    }
  }
  return 0;
}
```

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

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

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

#include <vector>
#include <exception>
#include <stdexcept>

#include "GuitarString.hpp"

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);

  // for (int i = 0; i < v.size(); i++) {
  //   cout << v.at(i) << endl;
  // }
  BOOST_REQUIRE_NO_THROW(GuitarString gs = GuitarString(v));

  cout << "Post BOOST_REQUIRE_NO_THROW line" << endl;
  GuitarString gs = GuitarString(v);

  // GS is 0 2000 4000 -10000
  BOOST_REQUIRE(gs.sample() == 0);
  cout << "check point 1" << endl;

  gs.tic();
  // it's now 2000 4000 -10000 996
  BOOST_REQUIRE(gs.sample() == 2000);
  cout << "check point 2" << endl;

  gs.tic();
  // it's now 4000 -10000 996 2988
  BOOST_REQUIRE(gs.sample() == 4000);
  cout << "check point 3" << endl;

  gs.tic();
  // it's now -10000 996 2988 -2988
  BOOST_REQUIRE(gs.sample() == -10000);
  cout << "check point 4" << endl;

  gs.tic();
  // it's now 996 2988 -2988 -4483
  BOOST_REQUIRE(gs.sample() == 996);
  cout << "check point 5" << endl;

  gs.tic();
  // it’s now 2988 -2988 -4483 1984
  BOOST_REQUIRE(gs.sample() == 2988);
  cout << "check point 6" << endl;

  gs.tic();
  // it’s now -2988 -4483 1984 0
  BOOST_REQUIRE(gs.sample() == -2988);
  cout << "check point 7" << endl;

  // 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);

  cout << "All tests passed!\n";
}
```

PS6: Airport

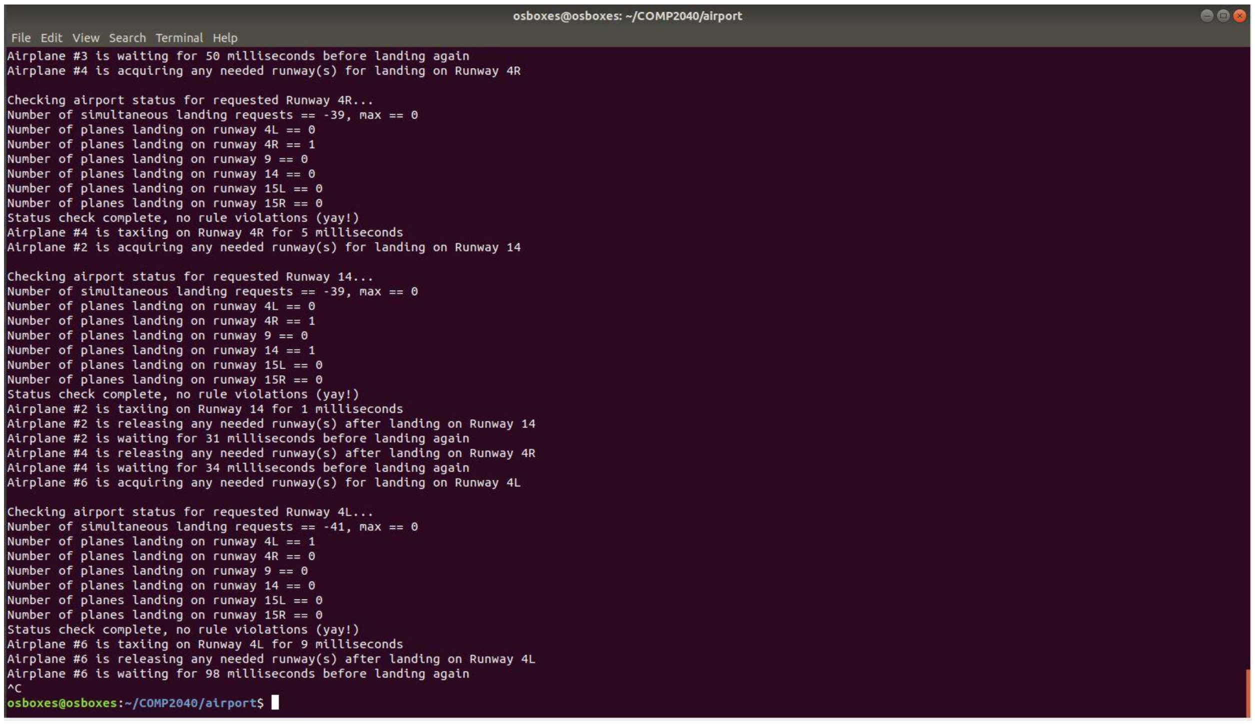
Assignment Overview:

This assignment was a simulation for planes landing at Logan Airport. It was mainly practice with multithreading and mutexes as most of the code had been written for us, we just had to implement the synchronization portion. We had to finish writing the functions inside AirportServer.cpp (as well as finish the AirportServer.h file) to close and reopen runways depending on whether or not a plane was currently using it or one that interfered with it. We only had to edit within the critical section of each function.

My implementation involved creating a large if/else block to lock/unlock the required runways. I started by locking the critical section with a unique lock, then telling a condition variable to make the mutex wait if the current number of landing request exceeded or was equal to the maximum number of requests.

What I Learned:

- How to use mutexes
- How to use lock/unlock for mutexes
- How to use condition variables


osboxes@osboxes: ~/COMP2040/airport
File Edit View Search Terminal Help

```
Airplane #3 is waiting for 50 milliseconds before landing again
Airplane #4 is acquiring any needed runway(s) for landing on Runway 4R

Checking airport status for requested Runway 4R...
Number of simultaneous landing requests == -39, max == 0
Number of planes landing on runway 4L == 0
Number of planes landing on runway 4R == 1
Number of planes landing on runway 9 == 0
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 #4 is taxiing on Runway 4R for 5 milliseconds
Airplane #2 is acquiring any needed runway(s) for landing on Runway 14

Checking airport status for requested Runway 14...
Number of simultaneous landing requests == -39, max == 0
Number of planes landing on runway 4L == 0
Number of planes landing on runway 4R == 1
Number of planes landing on runway 9 == 0
Number of planes landing on runway 14 == 1
Number of planes landing on runway 15L == 0
Number of planes landing on runway 15R == 0
Status check complete, no rule violations (yay!)
Airplane #2 is taxiing on Runway 14 for 1 milliseconds
Airplane #2 is releasing any needed runway(s) after landing on Runway 14
Airplane #2 is waiting for 31 milliseconds before landing again
Airplane #4 is releasing any needed runway(s) after landing on Runway 4R
Airplane #4 is waiting for 34 milliseconds before landing again
Airplane #6 is acquiring any needed runway(s) for landing on Runway 4L

Checking airport status for requested Runway 4L...
Number of simultaneous landing requests == -41, max == 0
Number of planes landing on runway 4L == 1
Number of planes landing on runway 4R == 0
Number of planes landing on runway 9 == 0
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 #6 is taxiing on Runway 4L for 9 milliseconds
Airplane #6 is releasing any needed runway(s) after landing on Runway 4L
Airplane #6 is waiting for 98 milliseconds before landing again
^C
osboxes@osboxes:~/COMP2040/airport$
```

This is a portion of the output from running the airport program

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

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

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

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

```
/**
*  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

```

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

#include "Airplane.h"

// The run() function in Airport will call this function
void Airplane::land()
{
        // obtain a seed from the system clock:
        unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();

        std::default_random_engine generator(seed);
        std::uniform_int_distribution<int> runwayNumberDistribution(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
```

```cpp
/**
*  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 violations of the rules of this simulation.
         */

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

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

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

}; // end class AirportRunways

#endif

```

```cpp
#include "AirportRunways.h"

int AirportRunways::runwayInUse[AirportRunways::NUM_RUNWAYS];

int AirportRunways::numLandingRequests = 0;

int AirportRunways::maxNumLandingRequests = 0;

mutex AirportRunways::checkMutex;


string AirportRunways::runwayName(RunwayNumber rn)
{
        switch (rn)
        {
        case RUNWAY_4L:
                return "4L";
        case RUNWAY_4R:
                return "4R";
        case RUNWAY_9:
                return "9";
        case RUNWAY_14:
                return "14";
        case RUNWAY_15L:
                return "15L";
        case RUNWAY_15R:
                return "15R";
        default:
                return "Unknown runway " + rn;
        } // end switch

} // end AirportRunways::runwayName()


 /**
  *  Check the status of the aiport with respect to any violation of the rules.
  */
void AirportRunways::checkAirportStatus(RunwayNumber requestedRunway)
{
        lock_guard<mutex> checkLock(checkMutex);

        bool crash = false; // Set to true if any rule is violated

        cout << "\nChecking airport status for requested Runway " << runwayName(requestedRunway) << "..." << endl;

        requestRunway(requestedRunway);

        // Check the number of landing requests
        cout << "Number of simultaneous landing requests == " << numLandingRequests
                 << ", max == " << maxNumLandingRequests << endl;

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

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

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

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

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

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

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

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

```cpp
/**
*  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

        mutex runway4LMutex;
        mutex runway4RMutex;
        mutex runway15LMutex;
        mutex runway15RMutex;
        mutex runway14Mutex;
        mutex runway9Mutex;

        mutex runwayMutex;

        //add a requestNum variable here, type int

        /**
         *  ***** Add declarations of your own Locks and
                  Condition Variables here *****
         */

        condition_variable _cv;

}; // end class AirportServer

#endif
```

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

#include "AirportServer.h"


/*
 *  Called by an Airplane when it wishes to land on a runway
 */




void AirportServer::reserveRunway(int airplaneNum, AirportRunways::RunwayNumber runway)
{
        // Acquire runway(s)
        {  // Begin critical region

          unique_lock<mutex> runwaysLock(runwaysMutex);

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

                        cout << "Airplane #" << airplaneNum << " is acquiring any needed runway(s) for landing on Runway "
                                 << AirportRunways::runwayName(runway) << endl;
                }

                /**
                 *  ***** Add your synchronization here! *****
                 */


                //lock mutexes in here

                _cv.wait(runwaysLock ,[]() { return AirportRunways::getNumLandingRequests() < 6;});
                //can condition on the number of requests currently active
                //If num of requests is greater than the Max number then wait, otherwise proceed
                //increase number of requests for each case


                if (AirportRunways::runwayName(runway) == "4L") {
                  //lock 4L, 15R, 15L
                  runway4LMutex.lock();
                  runway15RMutex.lock();
                  runway15LMutex.lock();
                } else if (AirportRunways::runwayName(runway) == "4R") {
                  //lock 4R, 9, 15R, 15L
                  runway4RMutex.lock();
                  runway9Mutex.lock();
                  runway15RMutex.lock();
                  runway15LMutex.lock();
                } else if (AirportRunways::runwayName(runway) == "15R") {
                  //lock 15R, 4L, 4R, 9
                  runway15RMutex.lock();
                  runway4LMutex.lock();
                  runway4RMutex.lock();
                  runway9Mutex.lock();
                } else if (AirportRunways::runwayName(runway) == "15L") {
                  //lock 15L, 4L, 4R
                  runway15LMutex.lock();
                  runway4LMutex.lock();
                  runway4RMutex.lock();
                } else if (AirportRunways::runwayName(runway) == "9") {
                  //lock 9, 4R, 15R
                  runway9Mutex.lock();
                  runway4RMutex.lock();
                  runway15RMutex.lock();
                } else if (AirportRunways::runwayName(runway) == "14") {
                  //lock 14
                  runway14Mutex.lock();
                }

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

                runwaysLock.unlock();

        } // End critical region

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

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

        {
                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

          //unique_lock<mutex> runwaysLock(runwaysMutex);

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

                        cout << "Airplane #" << airplaneNum << " is releasing any needed runway(s) after landing on Runway "
                                 << AirportRunways::runwayName(runway) << endl;
                }

                /**
                *  ***** Add your synchronization here! *****
                */

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

                if (AirportRunways::runwayName(runway) == "4L") {
                  //unlock 4L, 15R, 15L
                  runway4LMutex.unlock();
                  runway15RMutex.unlock();
                  runway15LMutex.unlock();
                } else if (AirportRunways::runwayName(runway) == "4R") {
                  //unlock 4R, 9, 15R, 15L
                  runway4RMutex.unlock();
                  runway9Mutex.unlock();
                  runway15RMutex.unlock();
                  runway15LMutex.unlock();
                } else if (AirportRunways::runwayName(runway) == "15R") {
                  //unlock 15R, 4L, 4R, 9
                  runway15RMutex.unlock();
                  runway4LMutex.unlock();
                  runway4RMutex.unlock();
                  runway9Mutex.unlock();
                } else if (AirportRunways::runwayName(runway) == "15L") {
                  //unlock 15L, 4L, 4R
                  runway15LMutex.unlock();
                  runway4LMutex.unlock();
                  runway4RMutex.unlock();
                } else if (AirportRunways::runwayName(runway) == "9") {
                  //unlock 9, 4R, 15R
                  runway9Mutex.unlock();
                  runway4RMutex.unlock();
                  runway15RMutex.unlock();
                } else if (AirportRunways::runwayName(runway) == "14") {
                  //unlock 14
                  runway14Mutex.unlock();
                }

                //decrease number of requests here
                // i.e. something like "numRequests--"
                AirportRunways::decNumLandingRequests();

                _cv.notify_one();

                //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 next landing for this Airplane
        std::uniform_int_distribution<int> waitTimeDistribution(1, MAX_WAIT_TIME);
        int waitTime = waitTimeDistribution(generator);

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

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

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

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

```cpp
/**
*  Airport driver program
*/

#include <iostream>
#include <thread>
#include <vector>

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

using namespace std;


// void run(Airplane* ap)
// {
//      ap->land();

// } // end run


int main(void)
{
        AirportServer as;

        vector<thread> apths; // Airplane threads

                                                  // Create and launch the individual Airplane threads
        for (int i = 1; i <= AirportRunways::NUM_AIRPLANES; i++)
        {
                Airplane* ap = new Airplane(i, &as);

                //apths.push_back(thread(&run, ap));
                apths.push_back(thread([ap]() { ap->land(); }));
        }

        // Wait for all Airplane threads to terminate (shouldn't happen!)
        for (auto& th : apths)
        {
                th.join();
        }

        return 0;

} // end main
```

PS7: Regular Expression Parsing

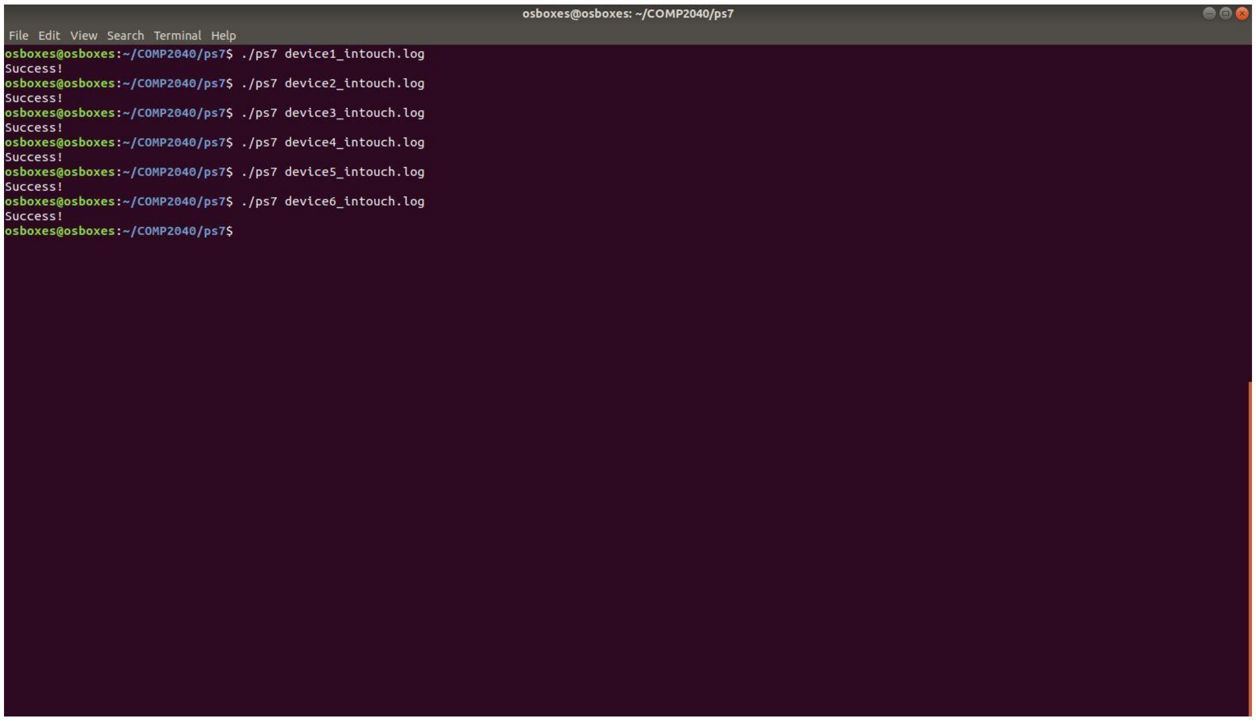
Assignment Overview:

This assignment was to parse through a boot log from the Kronos InTouch time clock. We were looking for a specific string that denoted a boot initiated and then another string that denoted a successful boot. Because a changing date was involved, we had to use regular expressions(regexes) to look for the strings. We also had to denote when a boot was unsuccessful. We had to report each boot and if it was successful (if so then the time taken as well) or not

My implementation created two regexes that were looking for the specific strings but were generalized to look for any type of date in front of the string. I parsed through the string looking for something that matched those strings. When I found them, I took the difference in time from the boot initiating to the boot finishing as the time taken for the boot. I used a Boolean value to keep track of whether or not a boot has been initiated. When the Boolean is false then a boot has completed (or hasn't happened yet), when the Boolean is true then a boot is currently happening. This is how I check for a failed boot, if a boot is trying to initiate while this value is true.

What I learned:

- How to use regexes
- How to use the boost time classes (Date and clock)
- More practice with using files for input


osboxes@osboxes: ~/COMP2040/ps7
File Edit View Search Terminal Help

```
osboxes@osboxes:~/COMP2040/ps7$ ./ps7 device1_intouch.log
Success!
osboxes@osboxes:~/COMP2040/ps7$ ./ps7 device2_intouch.log
Success!
osboxes@osboxes:~/COMP2040/ps7$ ./ps7 device3_intouch.log
Success!
osboxes@osboxes:~/COMP2040/ps7$ ./ps7 device4_intouch.log
Success!
osboxes@osboxes:~/COMP2040/ps7$ ./ps7 device5_intouch.log
Success!
osboxes@osboxes:~/COMP2040/ps7$ ./ps7 device6_intouch.log
Success!
osboxes@osboxes:~/COMP2040/ps7$
```

I made my program output "Success" on a successful finish

```
CC = g++
CFLAGS = -Wall -Werror -ansi -pedantic -std=c++0x
LBOOST = -lboost_regex -lboost_date_time

all: ps7

ps7: parse.o
	$(CC) parse.o -o ps7 $(LBOOST)

parse.o: parse.cpp
	$(CC) -c parse.cpp  $(CFLAGS)

clean:
	rm *.o
	rm ps7
```

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

  using namespace std;
  using namespace boost::gregorian;
  using namespace boost::posix_time;

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

  if (argc != 2) {
    cout << "Please enter with the form: ";
    cout << "./ps7 device1_intouch.log\n";
    return 0;
  }

  int lineNum = 1;
  string fileName(argv[1]);
  string line;
  ifstream logFile(fileName.c_str());
  bool bootStarted = false; //bool value to track if a boot sequence started

  string starting = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
  starting+= "([0-9]{2}):([0-9]{2}):([0-9]{2}): \\(log.c.166\\) server started";
  string ending = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";
  ending+= "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}):";
  ending+= "INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080";

  string bootMessage = "";

  //time related variables

  int hours, minutes, seconds;
  string startDate, endDate;
  string wholeDate;

  ptime startingTime;

  boost::regex startingRegex(starting);
  boost::regex endingRegex(ending);

  boost::smatch m;


  if (logFile.is_open()) {
    while(getline(logFile, line)) {
      //regex stuff

      if (boost::regex_search(line, m, startingRegex)) {

        startDate.clear();
        endDate.clear();
        //build a string for the staring date
        startDate = m[1] + "-" + m[2] + "-" + m[3] + " " + m[4]  + ":" + m[5] + ":" + m[6];
        wholeDate = m[1] + "-" + m[2] + "-" + m[3];
```

```cpp

     date startingDate(from_simple_string(wholeDate));

     //convert string times to int values
     hours = stoi(m[4]);
     minutes = stoi(m[5]);
     seconds = stoi(m[6]);

     startingTime = ptime(startingDate, time_duration(hours, minutes, seconds));

     //check if another boot started before previous finished
     if (bootStarted) {
      bootMessage += "**** Incomplete Boot ****\n\n";
     }

     //start building output message
     bootMessage += "===Device Boot===\n" + to_string(lineNum) + "(" + fileName + "): " + startDate + "Boot start\n";
     bootStarted = true; //true because a boot has started
    }

    //check if the boot completed
    if (regex_match(line, m, endingRegex)) {
     //build a string for the ending date
     endDate = m[1] + "-" + m[2] + "-" + m[3] + " " + m[4]  + ":" + m[5] + ":" + m[6];
     wholeDate = m[1] + "-" + m[2] + "-" + m[3];

     date endingDate(from_simple_string(wholeDate));

     //convert string times to int values
     hours = stoi(m[4]);
     minutes = stoi(m[5]);
     seconds = stoi(m[6]);

     ptime endingTime(endingDate, time_duration(hours, minutes, seconds));

     //calculate how long the boot took in milliseconds
     time_duration difference = endingTime - startingTime;

     //finish building output message
     bootMessage += to_string(lineNum) + "(" + fileName + "): " + endDate;
     bootMessage += "Boot Complete\n";
     bootMessage += "Boot Time: " + to_string(difference.total_milliseconds()) + "ms\n\n";

     bootStarted = false; //false because a boot ended
    }
    //increase line num counter
    lineNum++;
   }
   logFile.close();
  } else {
    cout << "FAILED TO OPEN FILE" << endl;
    return 0;
   }

  //create output .rpt file
  ofstream output(fileName + ".rpt");

  output << bootMessage;
  output.close();
  cout << "Success!\n";
  return 0;
}
```