Name: Ronney Sanchez

Date: 12/18/18

Course: COMP2040 Computing IV

Assignment: Project Portfolio

Fall 2018

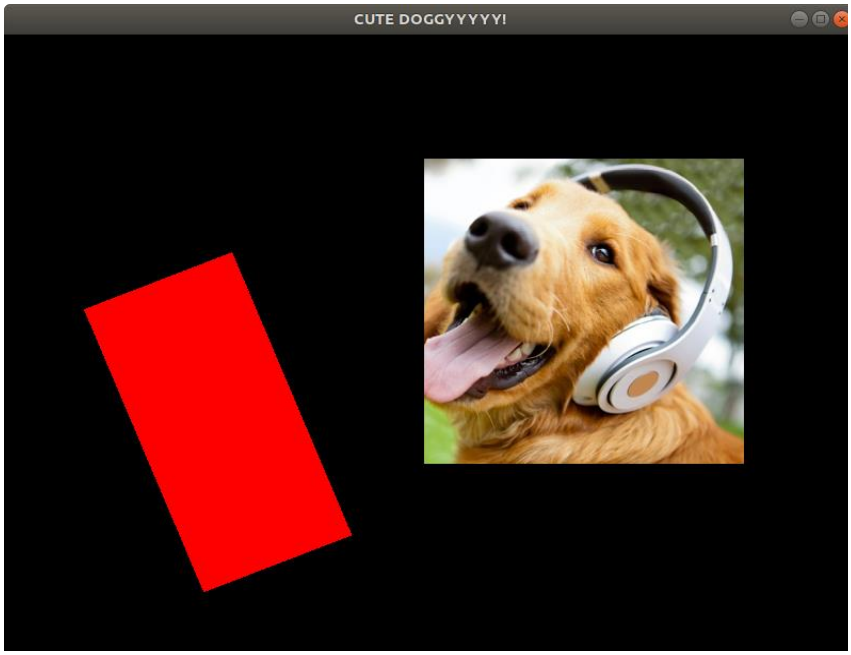
Table of Contents

PS0 Hello World with SFML

For this assignment I did a display of the SFML window. I used some cool features like a sprite image and a shape responding to keystrokes. For my sprite picture I used the key board strokes with the up, down, left, and right keys to move the picture. I added an additional feature to my SFML window. I changed the shape of the circle to a red rectangle. It also responds to the mouse key stroke. When I left click, the rectangle move diagonally down and rotate clockwise. When I right click, the rectangle move diagonally up and rotates counter-clockwise. I accomplished of how to animated sprites and shapes with movement according to keystrokes.

I encountered with some errors in my code. It involved with the shape and sprite functions for the SFML library. It was nothing big but it was just a slight error in which it is resolved and working properly.

~Pretty cool for shapes!



Makefile

```
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)
```

```
// File: main.cpp
/*Name: Ronney Sanchez
 *Course: COMP2040
 */
// Copyright 2018 Ronney Sanchez
#include <SFML/Graphics.hpp>
#include <iostream>

int main() {
    // Create the main window
    sf::RenderWindow window(sf::VideoMode(800, 800), "CUTE DOGGYYYYY!");
    sf::RectangleShape shape;
    shape.setSize(sf::Vector2f(300, 150));
    // Load a sprite to display
    sf::Texture texture;

    if (!texture.loadFromFile("sprite.jpg")) {
        std::cout << "Cannot open file!" << std::endl;
        return -1;
    }

    sf::Sprite sprite;
    sprite.setTexture(texture);
    sprite.setScale(0.5, 0.5);
    sprite.setPosition(200, 200);
    shape.setOutlineColor(sf::Color::Blue);
    shape.setFillColor(sf::Color::Red);

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

    // Clear window
    window.clear();
    // Draw the shape
    window.draw(shape);
    // Draw the sprite
    window.draw(sprite);

    if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
```

```
        shape.rotate(2);
        shape.move(1, 1);
    }

    if (sf::Mouse::isButtonPressed(sf::Mouse::Right)) {
        shape.rotate(-2);
        shape.move(-1, -1);
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
        sprite.move(-1, 0);
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
        sprite.move(1, 0);
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
        sprite.move(0, -1);
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
        sprite.move(0, 1);
    }

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

PS1 Recursive Graphics (Pythagoras tree)

The assignment required a lot of thinking towards math. I had to do some calculations of which spots in the SFML window I want to put my four vector points on. The math part of this assignment was the difficult part, but the implementation to draw was the easy part. I accomplish in solving the math of the shape implementation which is the difficult part and drawing the shape.

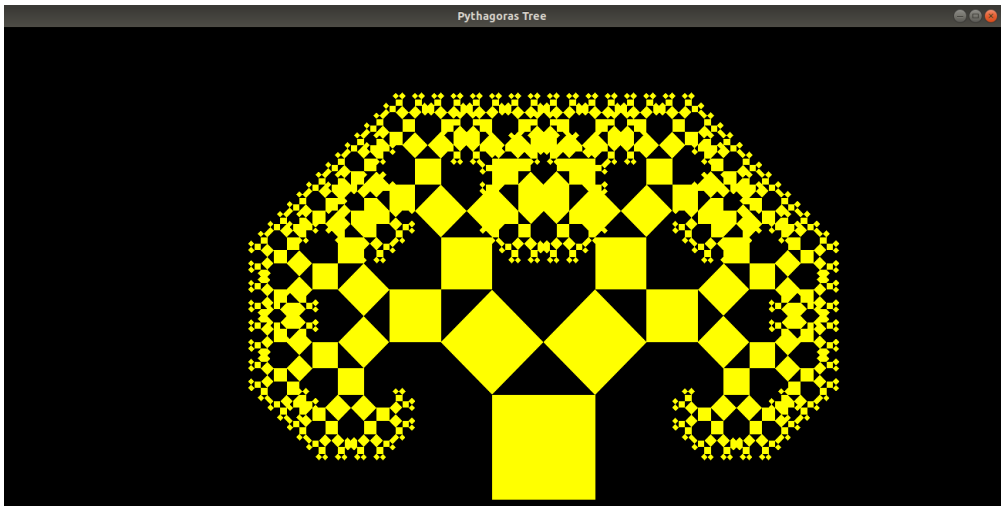
The key algorithms were trying to implement the base case for this assignment. I first implemented the four vector points for the square in the center of the SFML window. Then I took account of the triangle at the top of the square and started plotting my next two vector points for my new bases of the square. The idea of recursion is starting to play in.

The features I used were using a Convex Shape to draw my own square instead of using the Rectangle Shape class. I prefer plotting my own coordinates instead of the computer doing it for me because it is good practice. I also access the color library for the shape to provide some color filling effects. I only used one color for all squares because I am having trouble in using multiple colors.

I learned that programming requires a lot of math for using designs like SFML shapes and structures. The math is the most complicated idea in workings of code implementation but the graphics and the drawing is the easy part. I also learned that it is a struggle to get the program to work and requires a lot of time outside of class in doing this assignment.

One problem I encountered with is that when I started running the program, my computer froze. I noticed that it was a memory leak therefore I insert a destructor for my class to clean up

any leftover memories. Finally I got that issue resolved. My computer runs really well. Recursion is the most difficult idea to grasp.



Makefile

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

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

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

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

```
// File: PTree.hpp
/*
 *Name: Ronney Sanchez
 *Course: COMP2040 Computing IV
 *Assignment: PS1-Pythagoras Tree
 *
 */
// Copyright 2018 Ronney Sanchez
#ifndef PTree_INCLUDED
#define PTree_INCLUDED
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <iostream>
#include <cmath>

class PTree : public sf::Drawable {
 private:
        sf::Vector2f bottomLeft;
        sf::Vector2f bottomRight;
        sf::Vector2f topLeft;
        sf::Vector2f topRight;

        int initialDepth;

        PTree* child1;
        PTree* child2;
        void draw(sf::RenderTarget& target, sf::RenderStates states) const;

 public:
        PTree(int length, int depth);
        PTree(sf::Vector2f bLeft, sf::Vector2f bRight, int length, int depth);
        ~PTree();
};

#endif
```

```
//File: PTree.cpp
/*
 *Name: Ronney Sanchez
 *Course: COMP2040 Computing IV
 *Assignment: PS1-Pythagoras Tree
 */

// Copyright 2018 Ronney Sanchez
#include "PTree.hpp"

PTree::PTree(int length, int depth)
    : initialDepth(depth) {
    int newDepth = initialDepth - 1;

    bottomLeft.x = length/3.0;
    bottomLeft.y = length;

    bottomRight.x = (length/3.0) + (length/3.0);
    bottomRight.y = length;

    sf::Vector2f delta;
    sf::Vector2f nextPtr;

    delta.x = bottomRight.x - bottomLeft.x;
    delta.y = bottomLeft.y - bottomRight.y;

    topRight.x = bottomRight.x - delta.y;
    topRight.y = bottomRight.y - delta.x;

    topLeft.x = bottomLeft.x - delta.y;
    topLeft.y = bottomLeft.y - delta.x;

    nextPtr.x = topLeft.x + (delta.x - delta.y)/2.0;
    nextPtr.y = topLeft.y - (delta.x + delta.y)/2.0;

    if (newDepth == 0) {
        child1 = NULL;
        child2 = NULL;
    } else {
        child1 = new PTree(topLeft, nextPtr, length/2.0, newDepth);
        child2 = new PTree(nextPtr, topRight, length/2.0, newDepth);
    }
}
PTree::PTree(sf::Vector2f bLeft, sf::Vector2f bRight, int length, int depth)
    : initialDepth(depth) {
    int newDepth = initialDepth - 1;
```

```

        sf::Vector2f delta;
        sf::Vector2f nextPtr;

        bottomLeft = bLeft;
        bottomRight = bRight;

        delta.x = bRight.x - bLeft.x;
        delta.y = bLeft.y - bRight.y;

        topRight.x = bRight.x - delta.y;
        topRight.y = bRight.y - delta.x;

        topLeft.x = bLeft.x - delta.y;
        topLeft.y = bLeft.y - delta.x;

        nextPtr.x = topLeft.x + (delta.x - delta.y)/2.0;
        nextPtr.y = topLeft.y - (delta.x + delta.y)/2.0;

        if (newDepth == 0) {
            child1 = NULL;
            child2 = NULL;
        } else {
            child1 = new PTree(topLeft, nextPtr, length/2.0, newDepth);
            child2 = new PTree(nextPtr, topRight, length/2.0, newDepth);
        }
    }

    PTree::~PTree() {
        if (child1 != NULL) {
            delete (child1);
            delete (child2);
        }
    }

    void PTree::draw(sf::RenderTarget& target, sf::RenderStates states) const {
        sf::ConvexShape square;
        square.setPointCount(4);

        square.setPoint(0, topLeft);
        square.setPoint(1, topRight);
        square.setPoint(2, bottomRight);
        square.setPoint(3, bottomLeft);

        square.setFillColor(sf::Color::Green);
        square.move(500, 200);
```

```
        target.draw(square);

        if (child1 != NULL) {
            child1->draw(target, states);
            child2->draw(target, states);
        }
    }

    int main(int argc, char *argv[]) {
        if (argc < 3) {
            std::cout << "Pythagoras Tree [side-length][recursion-depth]" <<
                std::endl;
            return -1;
        }

        int side = atoi(argv[1]);
        int depth = atoi(argv[2]);

        sf::RenderWindow window(sf::VideoMode(6*side, 4*side), "Pythagoras Tree");

        PTree tree(side, depth);

        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;
}
```

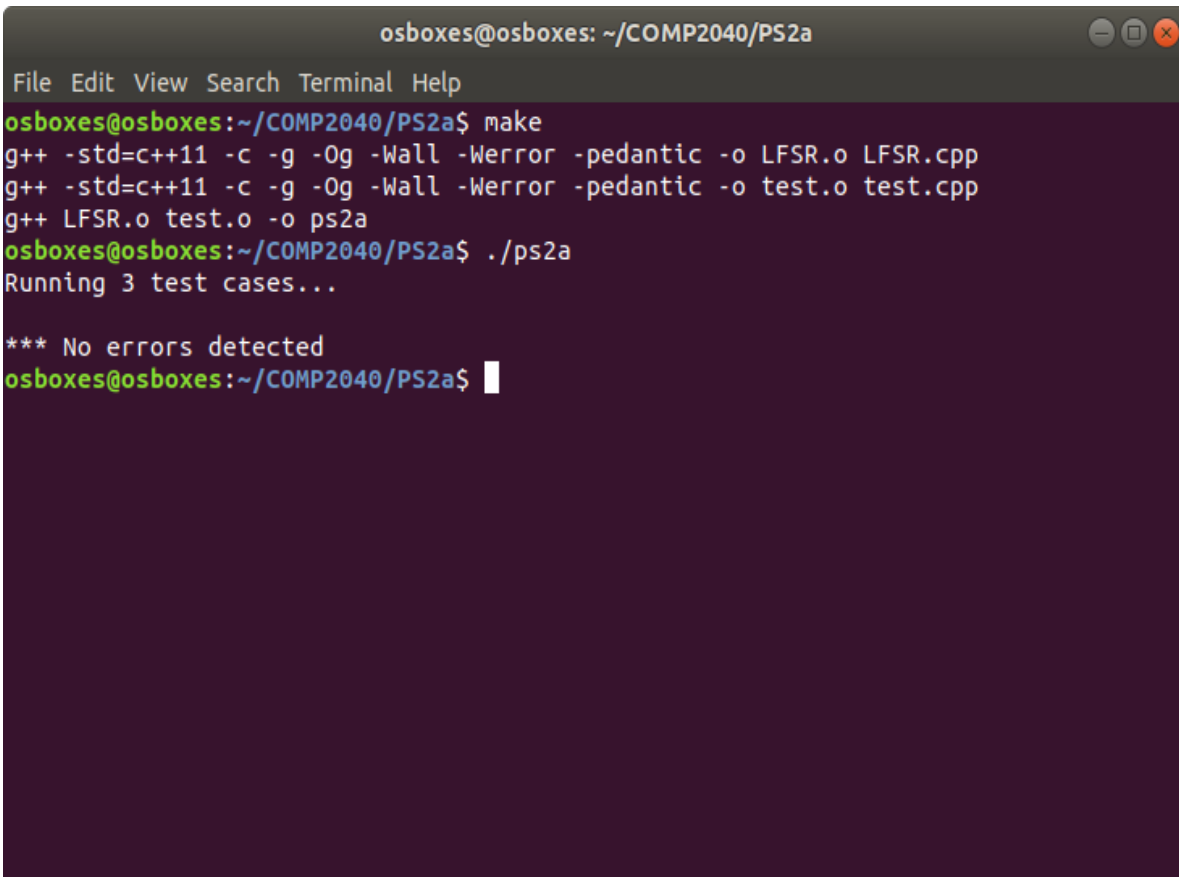
PS2 Linear Feedback Shift Register and Image Encoding

PS2a Linear Feedback Shift Register and Unit Testing

The representation I used for my register bits was a string of bits. I know that a string is an array of characters with a NULL terminator at the end. I used a string of bits with a maximum size of 32 as my seed and a tap integer as my tap bit location. I initialized the seed string and my tap position in my constructor.

I selected this kind of bit register because it is easier for me to do the string manipulation with the "erase" and "push_back" function. I then did a step function where it does the XOR operation between the left-most-bit and the tap bit. Since the seed is a string, I used the ".erase" and ".push_back" function to shift the string to the left with the new bit at the end and return the XOR result back to the user. I then did a generate function where it calls the step function k times and adds the results k times.

In my Boost testing file, I am testing two additional cases for my bit register. I am testing a small bit register and a 32 bit register with different tap position and different patterns of 0s and 1s. I am testing the expected output for each step function calls to determining if my generated output matches the expected output. I also do the same for both of them with the generate function I calculate the expected results on paper first, then I check with Boost to see if my generated output matches of what I have on paper. With my LFSR and Boost, it looks like my code is working properly without any errors on Boost. Boost is a very useful tool for unit testing.



Makefile

```
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
OBJ = LFSR.o test.o
DEPS =
LIBS =
EXE = ps2a

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

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

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

```
/* File: main.cpp
 * Name: Ronney Sanchez
 * Date: September26
 * Course: COMP2040 Computing IV
 * Assignment: PS2a
 * Email: Ronney_Sanchez@student.uml.edu
 */
// Copyright 2018 Ronney Sanchez

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

int main(int argc, char* argv[]) {
    if (argc != 3) {
        cout << "LFSR executable file [seed-string] [tap-position]" << endl;
        return -1;
    }

    string seed = argv[1];
    int tap = atoi(argv[2]);

    cout << "Bit pattern is " << seed << " with tap " << tap << " ." << endl;
    LFSR binaryString(seed, tap);

    cout << "OUTPUT FOR 10 STEPS" << endl;
    cout << "----------------------------------" <<  endl;
    for (int i = 0; i < 10; i++) {
            cout << binaryString << " " << binaryString.step() <<  endl;
    }

    cout << "\nOUTPUT FOR GENERATE 5 STEPS 10 TIMES" << endl;
    cout << "---------------------------------------" << endl;
    for (int i = 0; i < 10; i++) {
        cout << binaryString << " " << binaryString.generate(5) << endl;
    }
    return 0;
}
```

```
/* File: LFSR.hpp
 * Name: Ronney Sanchez
 * Date: September26
 * Course: COMP2040 Computing IV
 * Assignment: PS2a
 * Email: Ronney_Sanchez@student.uml.edu
 */
// Copyright 2018 Ronney Sanchez
#ifndef LFSR_INCLUDED
#define LFSR_INCLUDED
#include <iostream>
#include <string>

using namespace std;

class LFSR {
 public:
        LFSR(std::string seed, int t);
        int step();
        int generate(int k);
        friend ostream& operator<<(ostream &os, const LFSR &ws);


 private:
        string seed;
        int tap;
};

#endif
```

```
/* File: LFSR.cpp
 * Name: Ronney Sanchez
 * Date: September26
 * Course: COMP2040 Computing IV
 * Assignment: PS2a
 * Email: Ronney_Sanchez@student.uml.edu
 */
// Copyright 2018 Ronney Sanchez

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

LFSR::LFSR(string seed, int t) {
    for (unsigned int i = 0; i < seed.length(); i++) {
        while (seed.length() > 32) {
            cout << "Binary numbers only go up to 32 bits!" << endl;
            cout << "Enter a binary number: ";
            cin >> seed;
        }
        while (seed.at(i) != '0' && seed.at(i) != '1') {
            cout << "We're dealing only with binary numbers!" << endl;
            cout << "Enter a binary number: ";
            cin >> seed;
        }
    }
    this->seed = seed;
    this->tap = t;
}

int LFSR::step() {
    int target = seed.length() - tap - 1;
    char leftMostBit = seed.at(0);
    int bit = 0;

    if (seed.at(target) == leftMostBit) {
        bit = 0;
    } else {
        bit = 1;
    }

    char bitChar = '0';

    if (bit == 0) {
        bitChar = '0';
    } else {
        bitChar = '1';
```

```
    }
    seed.erase(seed.begin());
    seed.push_back(bitChar);
    return bit;
}

int LFSR::generate(int k) {
    int result = 0;
    for (int i = 0; i < k; i++) {
        int bitValue = step();
        result = (result * 2) + bitValue;
    }
    return result;
}

ostream& operator << (ostream &os, const LFSR &wr) {
    os << wr.seed;
    return os;
}
```

```
/*File: test.cpp
 * Name: Ronney Sanchez
 * Date: September26
 * Course: COMP2040 Computing IV
 * Assignment: PS2a
 * Email: Ronney_Sanchez@student.uml.edu
 */
// Copyright 2018 Ronney Sanchez
//
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/included/unit_test.hpp>
#include <iostream>
#include <string>
#include "./LFSR.hpp"

BOOST_AUTO_TEST_SUITE(Main)
/*This is a five bit string at tap two where it is going to check the
 * bit values for eight steps and determine if the bit values match.
 * Also it is going to check if the result of the generation of eight steps
 * matches.
 */
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);
}
/*This is a five bit string at tap three where it is going to check the
 * bit values for eight steps and determine if the bit values match.
 * Also it is going to check if the result of the generation of three steps
 * matches.
 */
BOOST_AUTO_TEST_CASE(fiveBitsTapAtThree) {
  LFSR l3("01011", 3);
  BOOST_REQUIRE(l3.step() == 1);
  BOOST_REQUIRE(l3.step() == 1);
  BOOST_REQUIRE(l3.step() == 1);
  BOOST_REQUIRE(l3.step() == 0);
```

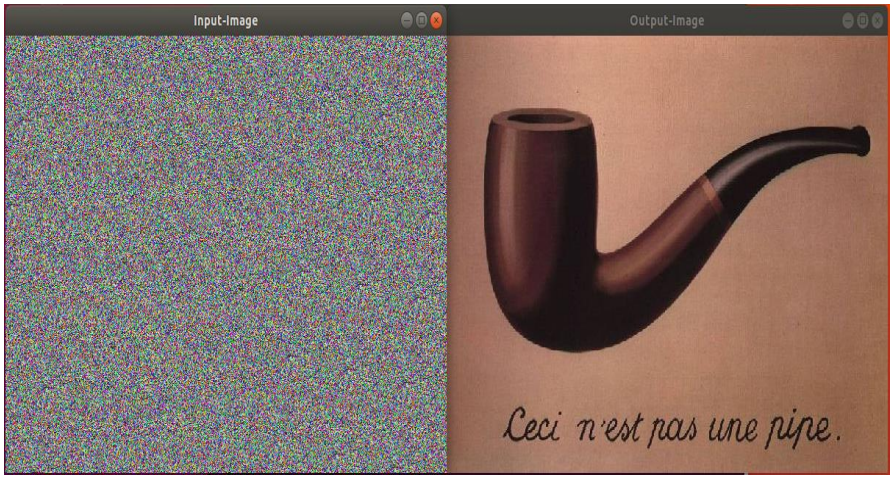
```
  BOOST_REQUIRE(l3.step() == 0);
  BOOST_REQUIRE(l3.step() == 0);
  BOOST_REQUIRE(l3.step() == 0);
  BOOST_REQUIRE(l3.step() == 1);
  LFSR l4("01011", 3);
  BOOST_REQUIRE(l4.generate(3) == 7);
}

/*This is a thirty-two bit string at tap two where it is going to check the
 * bit values for eight steps and determine if the bit values match.
 * Also it is going to check if the result of the generation of eight steps
 * matches.
 */
BOOST_AUTO_TEST_CASE(thirtyTwoBitsTapAtTwo) {
  LFSR l5("00110111000011111010101101001100", 2);
  BOOST_REQUIRE(l5.step() == 1);
  BOOST_REQUIRE(l5.step() == 0);
  BOOST_REQUIRE(l5.step() == 1);
  BOOST_REQUIRE(l5.step() == 0);
  BOOST_REQUIRE(l5.step() == 0);
  BOOST_REQUIRE(l5.step() == 0);
  BOOST_REQUIRE(l5.step() == 1);
  BOOST_REQUIRE(l5.step() == 1);
  LFSR l6("00110111000011111010101101001100", 2);
  BOOST_REQUIRE(l6.generate(8) == 163);
}
BOOST_AUTO_TEST_SUITE_END()
```

PS2b Encoding images with LFSR

In this assignment I learned how to use the LFSR class I made in PS2a and apply it to encrypt each pixels of an image using a specific seed string and a tap bit. My program works perfectly. It uses the LFSR class with a specific seed and tap bit to encrypt each pixel colors of the image. When I re-run the program again with the input and output image file swapped in the command line argument, but using the same seed string and tap bit, the image gets decrypted back to how it was in the beginning which is well implemented. My code completely work and got full points for this assignment.

Encoding Image

Decoding Image



Makefile

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

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

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

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

```
/*
 * Name: Ronney Sanchez
 * Date: October26
 * Course: COMP2040 Computing IV
 * Assignment: PS2b
 * Email: Ronney_Sanchez@student.uml.edu
 */
// Copyright 2018 Ronney Sanchez

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

int main(int argc, char *argv[]) {
    if (argc != 5) {
        cout << "PhotoMagic executable file [image source-file] [image encrypted-file] [seed bit
string] [tap position]" << endl;
        return -1;
    }
    string inputFile = argv[1];
    string outputFile = argv[2];
    string seed = argv[3];
    int tap = atoi(argv[4]);

    // LFSR object pointer
    LFSR *binaryString = new LFSR(seed, tap);

    sf::Image image;
    if (!image.loadFromFile(inputFile)) {
        cout << "ERROR: The file " << inputFile << " does not exist!" << endl;
        return -1;
    }

    sf::Texture texture1;
    texture1.loadFromImage(image);

    // p is a pixel
    sf::Color p;
    sf::Vector2u size = image.getSize();

    for (unsigned int x = 0; x <size.x; x++) {
        for (unsigned int y = 0; y < size.y; y++) {
            p = image.getPixel(x, y);
            p.r = p.r ^ binaryString->generate(90);
```

```
            p.g = p.g ^ binaryString->generate(70);
            p.b = p.b ^ binaryString->generate(150);
            image.setPixel(x, y, p);
        }
    }
    delete binaryString;

    sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Input-Image");
    sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "Output-Image");
    sf::Texture texture2;
    texture2.loadFromImage(image);

    sf::Sprite sprite1;
    sf::Sprite sprite2;
    sprite1.setTexture(texture1);
    sprite2.setTexture(texture2);

    while (window1.isOpen() && window2.isOpen()) {
        sf::Event event;
        while (window1.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window1.close();
            }
        }

        while (window2.pollEvent(event)) {
             if (event.type == sf::Event::Closed) {
                  window2.close();
             }
         }

         window1.clear();
         window1.draw(sprite1);
         window1.display();

         window2.clear();
         window2.draw(sprite2);
         window2.display();
    }

    if (!image.saveToFile(outputFile)) {
        cout << "ERROR: Cannot write to output file!" << endl;
        return -1;
    }
}
```

```
/* File: LFSR.hpp
 * Name: Ronney Sanchez
 * Date: September26
 * Course: COMP2040 Computing IV
 * Assignment: PS2a
 * Email: Ronney_Sanchez@student.uml.edu
 */
// Copyright 2018 Ronney Sanchez
#ifndef LFSR_INCLUDED
#define LFSR_INCLUDED
#include <iostream>
#include <string>

using namespace std;

class LFSR {
 public:
        LFSR(std::string seed, int t);
        int step();
        int generate(int k);
        friend ostream& operator<<(ostream &os, const LFSR &ws);


 private:
        string seed;
        int tap;
};

#endif

```

```
/* File: LFSR.cpp
 * Name: Ronney Sanchez
 * Date: September26
 * Course: COMP2040 Computing IV
 * Assignment: PS2a
 * Email: Ronney_Sanchez@student.uml.edu
 */
// Copyright 2018 Ronney Sanchez

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

LFSR::LFSR(string seed, int t) {
    for (unsigned int i = 0; i < seed.length(); i++) {
        while (seed.length() > 32) {
            cout << "Binary numbers only go up to 32 bits!" << endl;
            cout << "Enter a binary number: ";
            cin >> seed;
        }
        while (seed.at(i) != '0' && seed.at(i) != '1') {
            cout << "We're dealing only with binary numbers!" << endl;
            cout << "Enter a binary number: ";
            cin >> seed;
        }
    }
    this->seed = seed;
    this->tap = t;
}

int LFSR::step() {
    int target = seed.length() - tap - 1;
    char leftMostBit = seed.at(0);
    int bit = 0;

    if (seed.at(target) == leftMostBit) {
        bit = 0;
    } else {
        bit = 1;
    }

    char bitChar = '0';

    if (bit == 0) {
        bitChar = '0';
    } else {
        bitChar = '1';
```

```
    }
    seed.erase(seed.begin());
    seed.push_back(bitChar);
    return bit;
}

int LFSR::generate(int k) {
    int result = 0;
    for (int i = 0; i < k; i++) {
        int bitValue = step();
        result = (result * 2) + bitValue;
    }
    return result;
}

ostream& operator << (ostream &os, const LFSR &wr) {
    os << wr.seed;
    return os;
}
```

PS3 N-Body Simulation

PS3a Design a program that loads and displays a static universe

This assignment is a body simulation of the inner planets of our solar system. This assignment sets up the display of the sun with the 4 inner planets lined up to it. I accomplished in reading the text file correctly with the getline and input streams. I also accomplished in setting up the scale for the universe. Since the universe is very big, I had to divide the size into a smaller number for the SFML window.

I was happy that I was able to use smart pointers to access my planet images from their files because without smart pointers, the SFML window just displays white boxes because the reference from the texture of the file has been misplaced. Smart pointer takes care of those reference textures. I also accomplished in using shared pointers correctly.

One key algorithm that was central to the assignment is the use of smart pointers. We needed smart pointers to represent each celestial body of each planet and reference the textures of each planet.

Another key algorithm is using image, texture, and sprite in the Body class. We need those features to represents each of the planets. Also the position vectors were the important piece. It sets up each of the planets position to the right location of the window.

For my draw function in my Body class, I targeted my sprite with the states to display the planets drawing in the window. I overloaded the input stream operator to take every value for my position, velocity, mass, and filename, and store it to my input variable. I then return my input variable to the user. I basically read each row from the file.

My class does the right function for any arbitrary number of objects within the universe file. For my scaling, I took the size of the window and divided by a big e+ constant to take any arbitrary size and shrink it to an average window size.

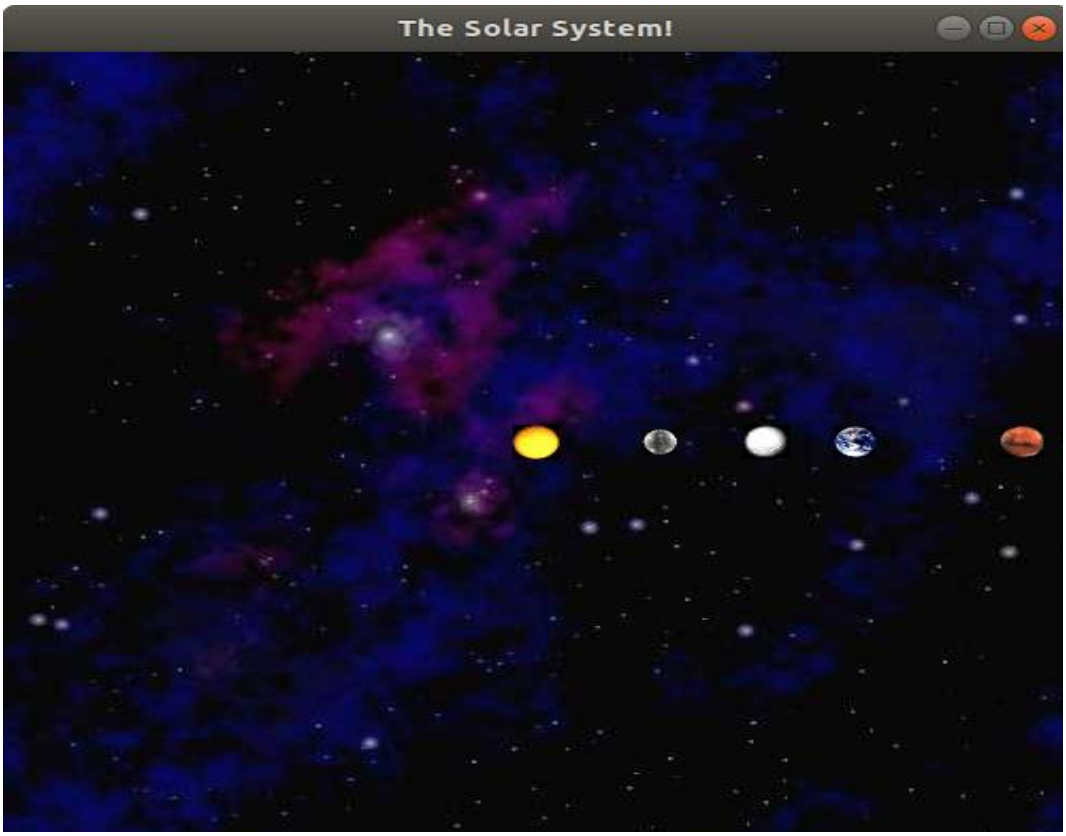
For the use of smart pointers, I used a vector of shared pointers of Body objects. I stored each shared pointers to the objects in a vector and used it for my SFML loop in a for loop for vector element accessing.

Yes, I complete the whole assignment successfully. My SFML window displays the universe correctly as mentioned in the assignment sheet. My smart pointers to the celestial bodies does the right function, my Body class do the right thing. I don't think that I have anything broken in this program.

I received help from a classmate. Patrick Fuller helped me try to read a line of text from a file by using getline and ifstream. He showed me the mechanics of using this input stream from a file and I does what I want it to do therefore I started using it to read from the universe file. Beside with the help of file reading, I did everything else by myself.

The problem I encountered with was that earlier, I was not able to display the planets to my SFML window. It showed up as white boxes. The reason was that the texture of those body objects lost its reference when it was re-positioned.

At that time I did not use smart pointers, but I figured out that smart pointers will fix the issue because smart pointers always keeps the reference of the body texture in place with the re-positioning. I then used smart pointers and my planets have been finally displayed.



Makefile

```
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
OBJ = Body.o main.o
DEPS =
LIBS = -lsfml-graphics -lsfml-window -lsfml-system
EXE = NBody

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

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

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

```
//File: main.cpp
/* Name: Ronney Sanchez
 * Date: October 24, 2018
 * Course: COMP2040 Computing 4
 * Assignment: PS3a
 */

#include <vector>
#include <memory>
#include "Body.hpp"

int main(int argc, char* argv[])
{
	if(argc != 1)
	{
		cout << "ERROR: execute file with < [filename] argument!" << endl;
	    return -1;
	}
	string xPos, yPos, xVelocity, yVelocity, mass, filename;

	unsigned int numParticles;
	float size;

	cin >> numParticles >> size;

	float windowSize = (size/5e+8);

	vector<shared_ptr<Body>> vecBody;
	for(unsigned int i = 0; i < numParticles; i++)
	{
		cin >> xPos >> yPos >> xVelocity >> yVelocity >> mass >> filename;
		auto body = make_shared<Body>(windowSize, xPos, yPos, xVelocity, yVelocity,
mass, filename);
		vecBody.push_back(body);

	}

	sf::RenderWindow window(sf::VideoMode(windowSize, windowSize), "The Solar
System!");

	sf::Image image;

	if(!image.loadFromFile("starfield.jpg"))
	{
		cerr << "ERROR: Unable to open \"starfield.jpg\"!" << endl;
		exit(1);
```

```
        }
        sf::Texture texture;
        texture.loadFromImage(image);
        sf::Sprite sprite;
        sprite.setTexture(texture);
        while(window.isOpen())
        {
        sf::Event event;

          while(window.pollEvent(event))
          {
                if(event.type == sf::Event::Closed)
                        {
                        window.close();
                        }
                }
          window.clear();
                window.draw(sprite);
                for(auto obj : vecBody)
                {
                        window.draw(*obj);
                }
                window.display();
        }
        return 0;
}
```

```
//File: Body.hpp
/* Name: Ronney Sanchez
 * Date: October 24, 2018
 * Course: COMP2040 Computing 4
 * Assignment: PS3a
 */

#ifndef Body_INCLUDED
#define Body_INCLUDED
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <iostream>
#include <string>
using namespace std;

class Body : public sf::Drawable
{
        public:
                Body(float size, string xPosition, string yPosition, string xVelocity, string
yVelocity, string myMass, string filename);

      private:
                sf::Image image;
                sf::Texture texture;
                sf::Sprite sprite;
                sf::Vector2f position;
                sf::Vector2f velocity;
                double mass;
                string filename;
                void draw(sf::RenderTarget& target, sf::RenderStates state) const;
                friend istream& operator>>(istream& in, Body& body);
};
#endif
```

```
//File: Body.cpp
/* Name: Ronney Sanchez
 * Date: October 24, 2018
 * Course: COMP2040 Computing 4
 * Assignment: PS3a
 */

#include "Body.hpp"

Body::Body(float size, string xPosition, string yPosition, string xVelocity, string yVelocity,
string myMass, string filename)
{
	position.x = stof(xPosition.c_str());
	position.y = stof(yPosition.c_str());

	velocity.x = stof(xVelocity.c_str());
	velocity.y = stof(yVelocity.c_str());

	mass = stof(myMass);

	this->filename = filename;

	image.loadFromFile(filename);

	texture.loadFromImage(image);

	sprite.setTexture(texture);

	sprite.setOrigin(sf::Vector2f(sprite.getLocalBounds().width,
sprite.getLocalBounds().height)/2.f);
	sprite.setPosition((size/2.f), (size/2.f));
	sprite.move((position.x/10e+8), (position.y/10e+8));
}

void Body::draw(sf::RenderTarget& target, sf::RenderStates state) const
{
	target.draw(sprite, state);
}

istream& operator>>(istream& in, Body& body)
{
	in >> body.position.x >> body.position.y >> body.velocity.x >> body.velocity.y >>
body.mass >> body.filename;
	body.image.loadFromFile(body.filename);
	body.texture.loadFromImage(body.image);
	body.sprite.setTexture(body.texture);
```

```
        body.sprite.setPosition((body.position.x), (body.position.y));
            return in;
    }
```

PS3b Using Newton's laws of physics, animate the universe

In the assignment I created the sun and four other planets within the universe. I made the planets rotate around the sun acting upon the force of the other planets and the gravity constant. My universe rotates counter-clockwise in a perfect circle around the sun. For my own universe, I replaced the sun with a blackhole and instead of using the 4 inner planets. I used the 4 outer planets which are from Jupiter to Neptune. However, the universe state does not start linear. I printed the final state of the universe from my regular simulation and wrote it to another text file and used that final state as my start state for my own universe. The planets start circular instead of linear.

I received help from Dr. Wilkes with the automatic file read input from the command line argument.

I also received help with my animation from Patrick Fuller (a classmate). He helped me with my calculations of the forces of each planet and the positioning of the universe.

The problem I encountered with was that at first, my planets were not rotating around the sun. My planets had its own rotation spot in the window and each planet was making random circle rotation around the window. The problem was that there was a simple mis-calculation in my class file, but I got it fixed.

The other issue I had was that I over wrote code at first because I was using a file input stream when I only just needed to overload the cin operator.

I was storing everything from a file pointer and using a vector to read text from the file pointer. I then went back and simplified it further with the overload cin operator.

Animation

Makefile

```
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
OBJ = Body.o main.o
DEPS =
LIBS = -lsfml-audio -lsfml-graphics -lsfml-window -lsfml-system
EXE = NBody

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

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

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

```
//File: main.cpp
/* Name: Ronney Sanchez
 * Date: November 7, 2018
 * Course: COMP2040 Computing 4
 * Assignment: PS3b
 */

#include <sstream>
#include <iomanip>
#include <vector>
#include <memory>
#include "Body.hpp"

int main(int argc, char* argv[])
{
	if(argc != 3)
	{
		cout << "ERROR: execute file with [Time], [Delta(Change in)-Time], <
[filename] argument!" << endl;
		return -1;
	}

	string  xPos, yPos, xVelocity, yVelocity, mass, filename;

	unsigned int numParticles;
	float size;
	float windowSize;
	double elapseTime = atof(argv[1]);
	double deltaT = atof(argv[2]);
	double time = 0.0;
	cout << "Elapse Time: "<< elapseTime << endl;
	cout << "Change In Time: " << deltaT << endl;

	cin >> numParticles >> size;
	windowSize = (size/5e+8);
	vector<shared_ptr<Body>> vecBody;
	for(unsigned int i = 0; i < numParticles; i++)
	{
		cin >> xPos >> yPos >> xVelocity >> yVelocity >> mass >> filename;
		auto body = make_shared<Body>(windowSize, xPos, yPos, xVelocity, yVelocity,
mass, filename);
		vecBody.push_back(body);

	}


```

```
	sf::RenderWindow window(sf::VideoMode(windowSize, windowSize), "The Solar System!");

	sf::Image image;

	if(!image.loadFromFile("starfield.jpg"))
	{
		cerr << "ERROR: Unable to open \"starfield.jpg\"!" << endl;
		exit(1);

	}

	sf::Texture texture;
	texture.loadFromImage(image);
	sf::Sprite sprite;
	sprite.setTexture(texture);

	sf::Music audio;

	if(!audio.openFromFile("2001.wav"))
	{
		cerr << "The file \"2001.wav\" does not exist!" << endl;
		exit(1);
	}

	audio.setVolume(5000);
	audio.play();
	while(window.isOpen() && time < elapseTime)
	{
	sf::Event event;

	   while(window.pollEvent(event))
	   {
		if(event.type == sf::Event::Closed || time >= elapseTime)
			{
			window.close();
			}
		}
	   window.clear();
		window.draw(sprite);

		sf::Font font;
		if(!font.loadFromFile("DIGITALDREAM.ttf"))
		{
			cerr << "The file \"DIGITALDREAM.ttf\" does not exist!" << endl;
			exit(1);
```

```
            }



            sf::Text clock;
            stringstream str;

            for(unsigned int i = 0; i < vecBody.size(); i++)
            {
                (*vecBody.at(i)).resetForce();
                for(unsigned int j = 0; j < vecBody.size(); j++)
                {
                    if(i != j)
                    {
                        (*vecBody.at(j)).resetForce();
                        (*vecBody.at(i)).addForce(*vecBody.at(j));
                    }
                }
                (*vecBody.at(i)).step(deltaT);
            window.draw(*vecBody.at(i));
                double timeClock = time;
                str << fixed << setprecision(2) << timeClock << " \n";
                string change;
                getline(str, change, '\n');
                clock.setString(change);
                clock.setFont(font);
                clock.setPosition(0, 0);
            }
            window.draw(clock);
            window.display();
            time += deltaT;
        }
        cout << "\nElapse Time: " << time << endl << endl;
        cout << "\nTHE UNIVERSE STATE" << endl;
        cout << numParticles << endl;
        cout << size << endl;
        for(auto obj : vecBody)
        {
            cout << *obj << endl;
        }
        return 0;
}
```

```
//File: Body.hpp
/* Name: Ronney Sanchez
 * Date: November 7, 2018
 * Course: COMP2040 Computing 4
 * Assignment: PS3b
 */

#ifndef Body_INCLUDED
#define Body_INCLUDED
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

class Body : public sf::Drawable
{
	public:
		Body(float size, string xPosition, string yPosition, string xVelocity, string
yVelocity, string myMass, string filename);
		void resetForce();
		void addForce(Body b);
		void step(double seconds);
		double distanceTo(Body b);

	private:
		const double G = 6.67e-11;
		sf::Image image;
		sf::Texture texture;
		sf::Sprite sprite;
		sf::Vector2f windowSize;
		sf::Vector2f position;
		sf::Vector2f velocity;
		sf::Vector2f acceleration;
		sf::Vector2f force;
		double mass;
		string filename;
		void draw(sf::RenderTarget& target, sf::RenderStates state) const;
		friend istream& operator>>(istream& in, Body& body);
		friend ostream& operator<<(ostream& out, const Body& body);
};
#endif
```

```
//File: Body.cpp
/* Name: Ronney Sanchez
 * Date: November 7, 2018
 * Course: COMP2040 Computing 4
 * Assignment: PS3b
 */

#include "Body.hpp"

Body::Body(float size, string xPosition, string yPosition, string xVelocity, string yVelocity,
string myMass, string filename)
{
	position.x = stof(xPosition.c_str());
	position.y = stof(yPosition.c_str());

	velocity.x = stof(xVelocity.c_str());
	velocity.y = stof(yVelocity.c_str());

	mass = stof(myMass);
	windowSize.x = size;
	windowSize.y = size;

	this->filename = filename;

	image.loadFromFile(filename);

	texture.loadFromImage(image);

	sprite.setTexture(texture);

	sprite.setOrigin(sf::Vector2f(sprite.getLocalBounds().width,
sprite.getLocalBounds().height)/2.f);
	sprite.setPosition((size/2.f), (size/2.f));
	sprite.move((position.x/10e+8), (position.y/10e+8));
}

void Body::draw(sf::RenderTarget& target, sf::RenderStates state) const
{
	target.draw(sprite, state);
}

void Body::resetForce()
{
	force.x = 0.0;
	force.y = 0.0;
}
```

```
void Body::addForce(Body b)
{
        double xDistance = b.position.x - this->position.x;
    double yDistance = b.position.y - this->position.y;
        double dist = sqrt(xDistance * xDistance + yDistance * yDistance);
        double F = (G * this->mass * b.mass) / (dist * dist);
        this->force.x += F * xDistance / dist;
        this->force.y += F * yDistance / dist;
}

void Body::step(double time)
{
    acceleration.x = force.x/mass;
    acceleration.y = force.y/mass;
    velocity.x += time * acceleration.x;
    velocity.y += time * acceleration.y;
    position.x += time * velocity.x;
    position.y += time * velocity.y;
        sprite.setPosition((position.x/10e+8), (position.y/10e+8) * -1);
        sprite.move((windowSize.x/2.0), (windowSize.y/2.0));
}

double Body::distanceTo(Body b)
{
    double xDistance = position.x - b.position.x;
    double yDistance = position.y - b.position.y;
    return sqrt((xDistance * xDistance) + (yDistance * yDistance));
}

istream& operator>>(istream& in, Body& body)
{
        in >> body.position.x >> body.position.y >> body.velocity.x >> body.velocity.y >>
body.mass >> body.filename;
        body.image.loadFromFile(body.filename);
    body.texture.loadFromImage(body.image);
    body.sprite.setTexture(body.texture);
    body.sprite.setPosition((body.position.x), (body.position.y));
        return in;
}

ostream& operator<<(ostream &out, const Body &body)
{
    out << body.position.x << " " << body.position.y << " " << body.velocity.x << " " <<
body.velocity.y << " " <<  body.mass << " " << body.filename;
    return out;
```

```
93      }
```

PS5 Ring Buffer and Guitar Hero

PS5a Ring Buffer with cpplint, testing, and exceptions

The assignment briefly creates a Ring Buffer and tests each functions to ensure that the Ring Buffer functions correctly. The assignment also uses the BOOST library to test multiple cases of the Ring Buffer.

I tested each of my functions for the Ring Buffer and it works properly. There were two ways of how I tested my functions. I tested it in main and also in the BOOST library. For my Ring Buffer I used a vector to store my values because vectors give me the flexibility to enqueue and dequeue values very easily with the push_back and erase function.

One key algorithm that was central to this assignment is that it was very similar to the Linear Feedback Shift Register assignment back in PS2. It looks like that this assignment uses the same structure like Linear Feedback Shift Register assignment but instead of implementing on an image, we are doing the RB implementation based on sound waves.

I implemented the enqueue function which just push_back new values to the back of the vector. I used dequeue to pop off values from the front. I also used a peek to get the value from the front of the vector. I also implemented the size() function to determine the size of the buffer. I also used exception handling to throw an exception to the user if they attempt to dequeue or peek an empty buffer or pushing more values to a full buffer.

I complete the whole assignment successfully. I tested my Ring Buffer both ways, in main and using the BOOST library. My enqueue and dequeue works really well by simply pushing new values to the back of the vector and popping values from the front. Also my size function works by returning the size of the vector. My isFull and isEmpty function works very well especially with my exception handling.

My implementation passes the unit test because I used up to 5 cases testing each of my function along with my exception handling. I used 5 cases to test for my enqueue, dequeue, peek, size, isFull, and isEmpty function.

Just to make sure that my exception handling works, I tried to go out of bounds with an enqueue on a full buffer and a dequeue on an empty buffer so that my exception handling catches that and it did. I used a run_time error exception handler to manage that.

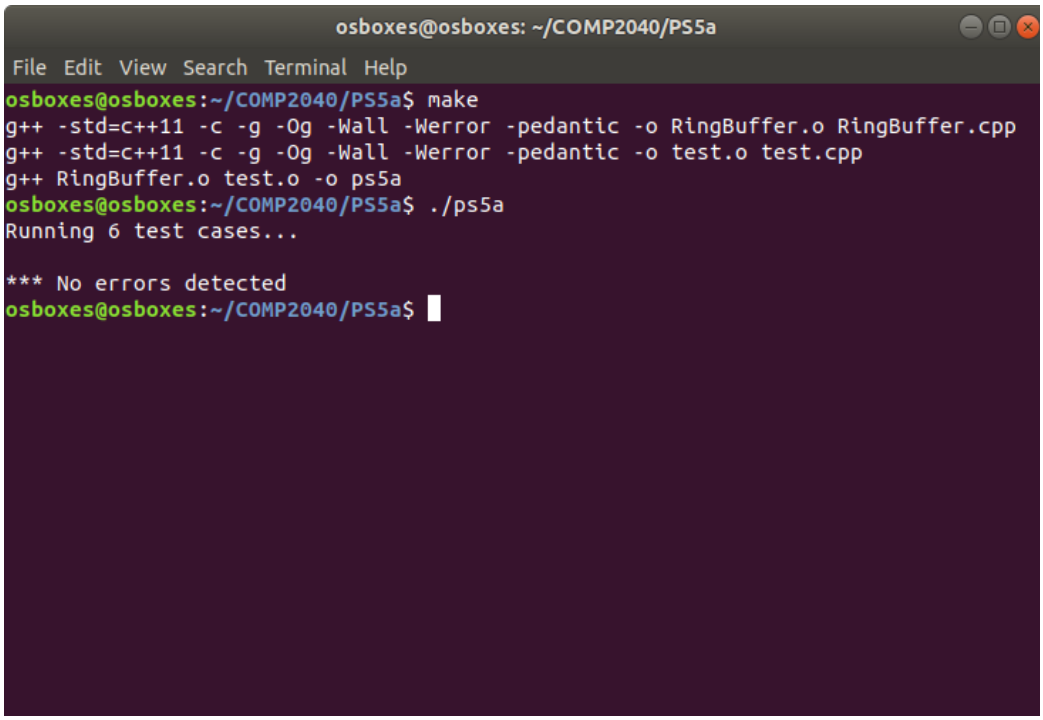
My performance of my RB implementation was kind of simple. I did not spent so long in getting the assignment done. I figured out that it was including a new library, implementing my Ring Buffer class, and simply testing each function with the unit test.

My unit test takes so long to compile because the BOOST library makes its own main function for each test cases and it takes some usual time to make a main function and implement the code for each test cases.

One problem that I encounter was that at some point during my unit testing, my isFull() function was returning 0 always even if the buffer was full. That was because I did not passed the capacity in the constructor argument to a member variable. I went back to fix that and my isFull() function started to work properly. Now it throws the error that the buffer is full even if it

is a vector because I handled the vector from increasing its size beyond the capacity. This assignment overall was very simple since it is just doing unit testing.

Output



Makefile

```
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
OBJ = RingBuffer.o test.o
DEPS =
LIBS =
EXE = ps5a

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

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

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

```
//File: main.cpp
/* Name: Ronney Sanchez
 * Course: COMP2040 Computing 4
 * Date: December 5, 2018
 * Assignment: PS5a
 */
// Copyright 2018 Ronney Sanchez

#include <iostream>
#include "RingBuffer.hpp"

int main(int argc, char* argv[]) {
    RingBuffer ringbuffer(5);
    std::cout << "Size: " << ringbuffer.size() << std::endl;
    std::cout << "Is Empty: " << ringbuffer.isEmpty() << std::endl;

    ringbuffer.enqueue(3);
    ringbuffer.enqueue(5);
    ringbuffer.enqueue(13);

    std::cout << "Size: " << ringbuffer.size() << std::endl;
    std::cout << "Is Full: " << ringbuffer.isFull() << std::endl;
    std::cout << "Is Empty: " << ringbuffer.isEmpty() << std::endl;

    ringbuffer.enqueue(25);
    ringbuffer.enqueue(11);

    std::cout << "Size: " << ringbuffer.size() << std::endl;
    std::cout << "Is Full: " << ringbuffer.isFull() << std::endl;

    std::cout << "Dequeue Value: " << ringbuffer.dequeue() << std::endl;
    std::cout << "Size: " << ringbuffer.size() << std::endl;

    std::cout << "Peek Value: " << ringbuffer.peek() << std::endl;
        std::cout << "Size: " << ringbuffer.size() << std::endl;
    std::cout << "Is Full: " << ringbuffer.isFull() << std::endl;

    return 0;
}
```

```
//File: RingBuffer.hpp
/* Name: Ronney Sanchez
 * Course: COMP2040 Computing 4
 * Date: December 5, 2018
 * Assignment: PS5a
 */

// Copyright 2018 Ronney Sanchez

#ifndef RINGBUFFER_INCLUDED
#define RINGBUFFER_INCLUDED
#include <stdint.h>
#include <iostream>
#include <vector>
#include <exception>
#include <stdexcept>

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

 private:
        int capacity;
        std::vector<int16_t> buffer;
        int front;
        int back;
        int numElements;
};
#endif
```

```
//File: RingBuffer.cpp
/* Name: Ronney Sanchez
 * Course: COMP2040 Computing 4
 * Date: December 5, 2018
 * Assignment: PS5a
 */
// Copyright 2018 Ronney Sanchez
#include "RingBuffer.hpp"

RingBuffer::RingBuffer(int capacity) {
    if (capacity < 1) {
        throw std::invalid_argument("ERROR: The capacity must be at least 1!");
    }
    this->capacity = capacity;
    front = 0;
    back = 0;
    numElements = 0;
}

RingBuffer::~RingBuffer() {}

int RingBuffer::size() {
    return buffer.size();
}

bool RingBuffer::isEmpty() {
    return (RingBuffer::size() == 0);
}

bool RingBuffer::isFull() {
    return (RingBuffer::size() == capacity);
}

void RingBuffer::enqueue(int16_t x) {
    if (RingBuffer::isFull()) {
        throw std::runtime_error("ERROR: The ring buffer is full!");
    }
    buffer.push_back(x);
    back = (back + 1) % capacity;
    numElements++;
}

int16_t RingBuffer::dequeue() {
    if (RingBuffer::isEmpty()) {
        throw std::runtime_error("ERROR: The ring buffer is empty!");
    }
```

```
    int16_t target = buffer.at(0);

    buffer.erase(buffer.begin());
    front = (front + 1) % capacity;
    numElements--;
    return target;
}

int16_t RingBuffer::peek() {
    if (RingBuffer::isEmpty()) {
        throw std::runtime_error("ERROR: The ring buffer is empty!");
    }
    return buffer.at(0);
}
```

```
// Name: Ronney Sanchez
// Course: COMP2040 Computing 4
// Date: December 5, 2018
// Assignment: PS5a

// Copyright 2015 fredm@cs.uml.edu for 91.204 Computing IV
// Wed Mar 25 06:32:17 2015

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

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

#include "RingBuffer.hpp"

BOOST_AUTO_TEST_SUITE(Main)
BOOST_AUTO_TEST_CASE(RBcontructor) {
    // normal constructor
    BOOST_REQUIRE_NO_THROW(RingBuffer(100));

    // this should fail
    BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
    BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
    RingBuffer ringbuffer(100);

    ringbuffer.enqueue(2);
    ringbuffer.enqueue(1);
    ringbuffer.enqueue(0);
    BOOST_REQUIRE(ringbuffer.dequeue() == 2);
    BOOST_REQUIRE(ringbuffer.dequeue() == 1);
    BOOST_REQUIRE(ringbuffer.dequeue() == 0);

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

BOOST_AUTO_TEST_CASE(RingBuffer_peek) {
    RingBuffer ringbuffer(100);

```

```
    ringbuffer.enqueue(5);
        ringbuffer.enqueue(9);
        ringbuffer.enqueue(12);
    ringbuffer.enqueue(7);

        BOOST_REQUIRE(ringbuffer.peek() == 5);
        BOOST_REQUIRE(ringbuffer.dequeue() == 5);
    BOOST_REQUIRE(ringbuffer.peek() == 9);
        BOOST_REQUIRE(ringbuffer.dequeue() == 9);
    BOOST_REQUIRE(ringbuffer.peek() == 12);
    BOOST_REQUIRE(ringbuffer.dequeue() == 12);
    BOOST_REQUIRE(ringbuffer.peek() == 7);
    BOOST_REQUIRE(ringbuffer.dequeue() == 7);

        BOOST_REQUIRE_THROW(ringbuffer.peek(), std::runtime_error);
}

BOOST_AUTO_TEST_CASE(RingBuffer_enqueue) {
        RingBuffer ringbuffer(5);

    BOOST_REQUIRE(ringbuffer.isFull() == 0);
    BOOST_REQUIRE(ringbuffer.isEmpty() == 1);

        ringbuffer.enqueue(1);
        ringbuffer.enqueue(4);
        ringbuffer.enqueue(7);
        ringbuffer.enqueue(13);
    ringbuffer.enqueue(32);

    BOOST_REQUIRE(ringbuffer.size() == 5);
    BOOST_REQUIRE(ringbuffer.isFull() == 1);

        BOOST_REQUIRE_THROW(ringbuffer.enqueue(67), std::runtime_error);
}

BOOST_AUTO_TEST_CASE(RingBuffer_size) {
        RingBuffer ringbuffer(100);

    BOOST_REQUIRE(ringbuffer.size() == 0);

        ringbuffer.enqueue(24);
        ringbuffer.enqueue(37);

    BOOST_REQUIRE(ringbuffer.size() == 2);

        ringbuffer.enqueue(11);
```

```
        ringbuffer.enqueue(19);

        BOOST_REQUIRE(ringbuffer.size() == 4);
        BOOST_REQUIRE(ringbuffer.peek() == 24);
        BOOST_REQUIRE(ringbuffer.dequeue() == 24);
        BOOST_REQUIRE(ringbuffer.size() == 3);
        BOOST_REQUIRE(ringbuffer.peek() == 37);
        BOOST_REQUIRE(ringbuffer.dequeue() == 37);
        BOOST_REQUIRE(ringbuffer.size() == 2);
        BOOST_REQUIRE(ringbuffer.peek() == 11);
        BOOST_REQUIRE(ringbuffer.dequeue() == 11);
        BOOST_REQUIRE(ringbuffer.size() == 1);
        BOOST_REQUIRE(ringbuffer.peek() == 19);
        BOOST_REQUIRE(ringbuffer.dequeue() == 19);
        BOOST_REQUIRE(ringbuffer.size() == 0);

        BOOST_REQUIRE_THROW(ringbuffer.peek(), std::runtime_error);
}

BOOST_AUTO_TEST_CASE(RingBuffer_isFull_isEmpty) {
        RingBuffer ringbuffer(3);

    BOOST_REQUIRE(ringbuffer.isEmpty() == 1);
    BOOST_REQUIRE(ringbuffer.isFull() == 0);

    ringbuffer.enqueue(1);
        ringbuffer.enqueue(4);
        ringbuffer.enqueue(7);

    BOOST_REQUIRE(ringbuffer.isEmpty() == 0);
        BOOST_REQUIRE(ringbuffer.isFull() == 1);
}

BOOST_AUTO_TEST_SUITE_END()
```

PS5b GuitarHero

I complete the whole assignment successfully. I got my window to display with different sounds on the keystroke of my keyboard. I also got my BOOST Test to work properly but in order to work, I have to insert a value on the Ring Buffer constructor and omit the value for my SFML Guitar implementation. My functions for my Guitar String class works because it passes the BOOST test without any errors.

I did not attempt any of the extra credit parts. I was very busy working on my main function and trying to get my SFML guitar implementation to work right. I did not have the chance to do the extra credit.

My Guitar String implementation passed the unit tests because I compiled it with my test file into an executable and all cases passed with no unexpected errors. However to pass the test, I need to push a random value to my Ring Buffer constructor because if it is empty, an unexpected error will occur.

I received help from my instructor (Dr. Tom Wilkes) and also one of my classmates. I received help from Patrick Fuller with my Guitar String implementation and also attempting to implement my main function.

I encountered a problem with my Guitar String unit test. I was receiving an unexpected error. An empty Ring Buffer exception handler was being thrown and I noticed that I am supposed to test my Guitar String with a non-empty Ring Buffer. I placed at least a value on the Ring Buffer and my error went away. I got my test to pass finally. I spent hours trying to get my main function to work and finally, it is done with different sounds implemented on each keystrokes.

This assignment was similar to PS2 with the Linear Feedback Shift Register but instead of an image file, we are doing the implementation with sound.

Makefile

```
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
OBJ = RingBuffer.o GuitarString.o GuitarHero.o
DEPS =
LIBS = -lsfml-audio -lsfml-graphics -lsfml-window -lsfml-system
EXE = GuitarHero

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

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

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

```
// File: GuitarHero.cpp
/* Name: Ronney Sanchez
 * Course: COMP2040 Computing 4
 * Date: December 14, 2018
 * Assignment: PS5b
 */

#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 "GuitarString.hpp"

int main() {
  std::vector<sf::SoundBuffer> vecSound;
  std::vector<std::vector<sf::Int16>> doubleVec;
  for(int i = 0; i < 37; i++)
  {
        double num = static_cast<double>(i);
        std::vector<sf::Int16> samples;
        GuitarString gs = GuitarString(num);
        gs.pluck();
        int duration = 8;

        for(int j = 0; j < SAMPLES_PER_SEC * duration; j++)
        {
              gs.tic();
              samples.push_back(gs.sample());
        }
        doubleVec.push_back(samples);
  }

  for(unsigned int i = 0; i < 37; i++)
  {
         sf::SoundBuffer soundbuffer;
```

```
            if(!soundbuffer.loadFromSamples(&doubleVec[i][0], doubleVec.size() - 1, 2,
SAMPLES_PER_SEC))
            {
                    throw std::runtime_error("ERROR: Failed to load from samples!");
            }
          vecSound.push_back(soundbuffer);
  }

  sf::Sound sound;
  std::vector<sf::Sound> soundStorage;

  for(unsigned int i = 0; i < 37; i++)
  {
            sound.setBuffer(vecSound.at(i));
            soundStorage.push_back(sound);
  }

  sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Guitar Hero");
  sf::Event event;

  while (window.isOpen()) {
    while (window.pollEvent(event)) {
      switch (event.type) {
      case sf::Event::Closed:
        window.close();
        break;

      case sf::Event::KeyPressed:
        switch (event.key.code) {
        case sf::Keyboard::Q:
          soundStorage.at(0).play();
          break;

        case sf::Keyboard::Num2:
          soundStorage.at(1).play();
          break;

            case sf::Keyboard::W:
          soundStorage.at(2).play();
          break;

            case sf::Keyboard::E:
          soundStorage.at(3).play();
          break;

            case sf::Keyboard::Num4:
```

```
            soundStorage.at(4).play();
            break;

              case sf::Keyboard::R:
            soundStorage.at(5).play();
            break;

              case sf::Keyboard::Num5:
            soundStorage.at(6).play();
            break;

              case sf::Keyboard::T:
            soundStorage.at(7).play();
            break;

              case sf::Keyboard::Y:
            soundStorage.at(8).play();
            break;

              case sf::Keyboard::Num7:
            soundStorage.at(9).play();
            break;

              case sf::Keyboard::U:
            soundStorage.at(10).play();
            break;

              case sf::Keyboard::Num8:
            soundStorage.at(11).play();
            break;

              case sf::Keyboard::I:
            soundStorage.at(12).play();
            break;

              case sf::Keyboard::Num9:
            soundStorage.at(13).play();
            break;

              case sf::Keyboard::O:
            soundStorage.at(14).play();
            break;

              case sf::Keyboard::Num0:
            soundStorage.at(15).play();
            break;
```

```

            case sf::Keyboard::P:
        soundStorage.at(16).play();
        break;

            case sf::Keyboard::LBracket:
        soundStorage.at(17).play();
        break;

            case sf::Keyboard::Equal:
        soundStorage.at(18).play();
        break;

            case sf::Keyboard::Z:
        soundStorage.at(19).play();
        break;

            case sf::Keyboard::X:
        soundStorage.at(20).play();
        break;

            case sf::Keyboard::D:
        soundStorage.at(21).play();
        break;

            case sf::Keyboard::C:
        soundStorage.at(22).play();
        break;

            case sf::Keyboard::F:
        soundStorage.at(23).play();
        break;

            case sf::Keyboard::V:
        soundStorage.at(24).play();
        break;

            case sf::Keyboard::G:
        soundStorage.at(25).play();
        break;

            case sf::Keyboard::B:
        soundStorage.at(26).play();
        break;

            case sf::Keyboard::N:
```

```
            soundStorage.at(27).play();
            break;

              case sf::Keyboard::J:
            soundStorage.at(28).play();
            break;

              case sf::Keyboard::M:
            soundStorage.at(29).play();
            break;

              case sf::Keyboard::K:
            soundStorage.at(30).play();
            break;

              case sf::Keyboard::Comma:
            soundStorage.at(31).play();
            break;

              case sf::Keyboard::Period:
            soundStorage.at(32).play();
            break;

              case sf::Keyboard::SemiColon:
            soundStorage.at(33).play();
            break;

              case sf::Keyboard::Slash:
            soundStorage.at(34).play();
            break;

              case sf::Keyboard::Quote:
            soundStorage.at(35).play();
            break;

              case sf::Keyboard::Space:
            soundStorage.at(1).play();
            break;

          default:
            break;
          }

        default:
          break;
        }
```

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

```
//File: GuitarString.hpp
/* Name: Ronney Sanchez
 * Course: COMP2040 Computing 4
 * Date: December 14, 2018
 * Assignment: PS5b
 */
// Copyright 2018 Ronney Sanchez

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

#include <cmath>
#include <limits.h>

#include "RingBuffer.hpp"

#define FREQ_CONST 440.0
#define CONCERT_A 220.0
#define SAMPLES_PER_SEC 44100
#define DECAY_FACTOR 0.996

class GuitarString {
	public:
		explicit GuitarString(double frequency);
		explicit GuitarString(std::vector<sf::Int16> init);
		GuitarString(const GuitarString &obj) {};
			// no copy constructor

		~GuitarString();
		void pluck();
		void tic();
		sf::Int16 sample();
		int time();

	private:
		RingBuffer* _rb;
		int _time;
		int bufferSize;
};
#endif
```

```
// File: GuitarString.cpp
/* Name: Ronney Sanchez
 * Course: COMP2040 Computing 4
 * Date: December 14, 2018
 * Assignment: PS5b
 */

#include "GuitarString.hpp"

GuitarString::GuitarString(double frequency) {
	bufferSize = ceil(FREQ_CONST  * pow(2, (frequency - 24)/12.0));
	_rb = new RingBuffer(bufferSize);
}

GuitarString::GuitarString(std::vector<sf::Int16> init) {
	_rb = new RingBuffer(init.size());
	(*_rb).dequeue();

	for(unsigned int i = 0; i < init.size(); i++)
	{
		(*_rb).enqueue(init.at(i));
	}

}

GuitarString::~GuitarString() {
	delete _rb;
}

void GuitarString::pluck() {
	if((*_rb).isFull())
	{
		while(!(*_rb).isEmpty())
		{
			(*_rb).dequeue();
		}
	}

	for(int i = 0; i < bufferSize; i++)
	{
		int16_t num = (rand() % 65535) - 32768;
		(*_rb).enqueue(num);
	}

	(*_rb).dequeue();

```

```
}

void GuitarString::tic() {
	int16_t karplus = DECAY_FACTOR * (0.5 * ((*_rb).dequeue() + (*_rb).peek()));
	(*_rb).enqueue(karplus);
	_time++;
}

sf::Int16 GuitarString::sample() {
	return (*_rb).peek();
}

int GuitarString::time() {
	return _time;
}
```

```
//File: RingBuffer.hpp
/* Name: Ronney Sanchez
 * Course: COMP2040 Computing 4
 * Date: December 14, 2018
 * Assignment: PS5b
 */

// Copyright 2018 Ronney Sanchez

#ifndef RINGBUFFER_INCLUDED
#define RINGBUFFER_INCLUDED
#include <stdint.h>
#include <iostream>
#include <vector>
#include <exception>
#include <stdexcept>

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

 private:
        int capacity;
        std::vector<int16_t> buffer;
        int front;
        int back;
        int numElements;
};
#endif
```

```
//File: RingBuffer.cpp
/* Name: Ronney Sanchez
 * Course: COMP2040 Computing 4
 * Date: December 14, 2018
 * Assignment: PS5b
 */
// Copyright 2018 Ronney Sanchez
#include "RingBuffer.hpp"

RingBuffer::RingBuffer(int capacity) {
    if (capacity < 1) {
        throw std::invalid_argument("ERROR: The capacity must be at least 1!");
    }
    this->capacity = capacity;
    front = 0;
    back = 0;
    numElements = 0;
}

RingBuffer::~RingBuffer() {}

int RingBuffer::size() {
    return buffer.size();
}

bool RingBuffer::isEmpty() {
    return (RingBuffer::size() == 0);
}

bool RingBuffer::isFull() {
    return (RingBuffer::size() == capacity);
}

void RingBuffer::enqueue(int16_t x) {
    if (RingBuffer::isFull()) {
        throw std::runtime_error("ERROR: The ring buffer is full!");
    }
    buffer.push_back(x);
    back = (back + 1) % capacity;
    numElements++;
}

int16_t RingBuffer::dequeue() {
    if (RingBuffer::isEmpty()) {
        throw std::runtime_error("ERROR: The ring buffer is empty!");
    }
```

```
    int16_t target = buffer.at(0);

    buffer.erase(buffer.begin());
    front = (front + 1) % capacity;
    numElements--;
    return target;
}

int16_t RingBuffer::peek() {
    if (RingBuffer::isEmpty()) {
        throw std::runtime_error("ERROR: The ring buffer is empty!");
    }
    return buffer.at(0);
}
```

Airport

The assignment is an airport simulation which requests up to a total of six planes to land and uses a mutual lock to handle each landing requests. The program specifies a certain plane to land at a certain path or specifies more than one plane to land at non-conflicting paths. The program uses mutual locks and condition variables to handle the landing request.

One key algorithm that was central to this assignment is concurrency. This assignment is related to the Operating System course. This assignment has a lot of Operating System knowledge involved. This assignment also has deadlocks and starvation problem with the airplane landing.

Another key algorithm that is central is mutual exclusion. If we want more than one plane to land then we need to tell the program to find non-conflicting paths for these two planes to land other-wise, one plane must wait while the other plane finish landing to clear the runway for the next plane to land. Then when more requests comes, those plane request must be put to sleep until the other requests has been resolved.

I added more mutual lock and condition variables to the program to handle the landing requests to avoid crashes. I also modified the landing request and release functions to handle any conflicting landing paths. Each mutex variables are for each runway paths. I used switch statements to handle the locks and unlocks of each runways.

I learned a little more about Operating System with this assignment because this assignment uses a lot of concurrency knowledge which involves mutual exclusion, locks, unlocks, condition variables, deadlocks, and starvation. I learned that we can control variables using the mutual lock variables.

I received help from the instructor (Dr. Tom Wilkes) and Patrick Fuller with this assignment. Even though I received help, I was still struggling to get this assignment working in which I even worked on this over the Thanskgiving break. Finally I achieved where the program was going wrong and it is fixed. My program finally works for 15+ minutes as required with no more than six landing requests and more than one plane landing at non-conflicting paths. I was encountering crashes in my airport simulation. I kept debugging and got the program to run for a couple of seconds more before another crash happens. I was playing around with the mutual exclusion variables to note which spots in the program it needs to be in. I noticed that when the number of landing requests reached more than 1 the more likely the program will crash. I used switch statements to lock and unlock the mutex variables for each paths being occupied and paths that intersect the occupied paths. Finally my program is stable enough to control the landing request safely for each airplane. This program is probably not difficult but we need to know the right method of controlling the lock and unlock variables with the condition variables.

Output

```
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 == 1, max == 1
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 4 microseconds
Airplane #6 is acquiring any needed runway(s) for landing on Runway 15R
Airplane #4 is releasing any needed runway(s) after landing on Runway 4R

Checking airport status for requested Runway 15R...
Number of simultaneous landing requests == 2, max == 2
Number of planes landing on runway 4L == 0
```

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 == 1
Status check complete, no rule violations (yay!)
Airplane #6 is taxiing on Runway 15R for 9 microseconds
Airplane #4 is waiting for 20 microseconds before landing again
Airplane #1 is acquiring any needed runway(s) for landing on Runway 4R
Airplane #6 is releasing any needed runway(s) after landing on Runway 15R
Airplane #2 is acquiring any needed runway(s) for landing on Runway 14

Checking airport status for requested Runway 4R...
Number of simultaneous landing requests == 2, max == 2
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 #1 is taxiing on Runway 4R for 7 microseconds
Airplane #6 is waiting for 87 microseconds before landing again

Checking airport status for requested Runway 14...
Number of simultaneous landing requests == 2, max == 2
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 6 microseconds
Airplane #1 is releasing any needed runway(s) after landing on Runway 4R
Airplane #1 is waiting for 99 microseconds before landing again
Airplane #2 is releasing any needed runway(s) after landing on Runway 14
Airplane #2 is waiting for 54 microseconds before landing again
Airplane #7 is acquiring any needed runway(s) for landing on Runway 15R

Checking airport status for requested Runway 15R...
Number of simultaneous landing requests == 1, max == 2
Number of planes landing on runway 4L == 0
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 == 1
Status check complete, no rule violations (yay!)
Airplane #7 is taxiing on Runway 15R for 2 microseconds
Airplane #5 is acquiring any needed runway(s) for landing on Runway 15L
Airplane #7 is releasing any needed runway(s) after landing on Runway 15R

Checking airport status for requested Runway 15L...
Number of simultaneous landing requests == 2, max == 2
Number of planes landing on runway 4L == 0
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 == 1
Number of planes landing on runway 15R == 0
Status check complete, no rule violations (yay!)
Airplane #5 is taxiing on Runway 15L for 3 microseconds
Airplane #7 is waiting for 35 microseconds before landing again
Airplane #3 is acquiring any needed runway(s) for landing on Runway 15L
Airplane #5 is releasing any needed runway(s) after landing on Runway 15L
Airplane #5 is waiting for 92 microseconds before landing again

Checking airport status for requested Runway 15L...
Number of simultaneous landing requests == 1, max == 2
Number of planes landing on runway 4L == 0
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 == 1
Number of planes landing on runway 15R == 0
Status check complete, no rule violations (yay!)
Airplane #3 is taxiing on Runway 15L for 1 microseconds
Airplane #3 is releasing any needed runway(s) after landing on Runway 15L
Airplane #3 is waiting for 63 microseconds 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 == 1, max == 2
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 4 microseconds
```

```
Airplane #1 is acquiring any needed runway(s) for landing on Runway 4R
Airplane #2 is acquiring any needed runway(s) for landing on Runway 14
Airplane #6 is acquiring any needed runway(s) for landing on Runway 14
Airplane #4 is releasing any needed runway(s) after landing on Runway 4R

Checking airport status for requested Runway 4R...
Number of simultaneous landing requests == 2, max == 2
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 #1 is taxiing on Runway 4R for 3 microseconds

Checking airport status for requested Runway 14...
Number of simultaneous landing requests == 3, max == 3
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 8 microseconds
Airplane #5 is acquiring any needed runway(s) for landing on Runway 15R
Airplane #7 is acquiring any needed runway(s) for landing on Runway 15R
Airplane #1 is releasing any needed runway(s) after landing on Runway 4R
Airplane #2 is releasing any needed runway(s) after landing on Runway 14

Checking airport status for requested Runway 15R...
Number of simultaneous landing requests == 4, max == 4
Number of planes landing on runway 4L == 0
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 == 1
Status check complete, no rule violations (yay!)
Airplane #5 is taxiing on Runway 15R for 8 microseconds
Airplane #4 is waiting for 40 microseconds before landing again

Checking airport status for requested Runway 14...
Number of simultaneous landing requests == 4, max == 4
Number of planes landing on runway 4L == 0
```

```
Number of planes landing on runway 4R == 0
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 == 1
Status check complete, no rule violations (yay!)
Airplane #6 is taxiing on Runway 14 for 1 microseconds
Airplane #5 is releasing any needed runway(s) after landing on Runway 15R
Airplane #5 is waiting for 67 microseconds before landing again

Checking airport status for requested Runway 15R...
Number of simultaneous landing requests == 4, max == 4
Number of planes landing on runway 4L == 0
Number of planes landing on runway 4R == 0
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 == 1
Status check complete, no rule violations (yay!)
Airplane #7 is taxiing on Runway 15R for 10 microseconds
Airplane #1 is waiting for 35 microseconds before landing again
Airplane #2 is waiting for 47 microseconds before landing again
Airplane #6 is releasing any needed runway(s) after landing on Runway 14
Airplane #6 is waiting for 77 microseconds before landing again
Airplane #7 is releasing any needed runway(s) after landing on Runway 15R
Airplane #7 is waiting for 36 microseconds before landing again
Airplane #3 is acquiring any needed runway(s) for landing on Runway 15R

Checking airport status for requested Runway 15R...
Number of simultaneous landing requests == 1, max == 4
Number of planes landing on runway 4L == 0
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 == 1
Status check complete, no rule violations (yay!)
Airplane #3 is taxiing on Runway 15R for 4 microseconds
Airplane #3 is releasing any needed runway(s) after landing on Runway 15R
Airplane #3 is waiting for 55 microseconds before landing again
Airplane #4 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 == 4
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 #4 is taxiing on Runway 9 for 1 microseconds
Airplane #5 is acquiring any needed runway(s) for landing on Runway 14

Checking airport status for requested Runway 14...
Number of simultaneous landing requests == 2, max == 4
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 == 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 #5 is taxiing on Runway 14 for 3 microseconds
Airplane #2 is acquiring any needed runway(s) for landing on Runway 14
Airplane #1 is acquiring any needed runway(s) for landing on Runway 9
Airplane #5 is releasing any needed runway(s) after landing on Runway 14
Airplane #4 is releasing any needed runway(s) after landing on Runway 9

Checking airport status for requested Runway 14...
Number of simultaneous landing requests == 3, max == 4
Number of planes landing on runway 4L == 0
Number of planes landing on runway 4R == 0
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 10 microseconds

Checking airport status for requested Runway 9...
Number of simultaneous landing requests == 4, max == 4
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 == 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 #1 is taxiing on Runway 9 for 5 microseconds
Airplane #1 is releasing any needed runway(s) after landing on Runway 9
Airplane #4 is waiting for 41 microseconds before landing again
```

```
Airplane #2 is releasing any needed runway(s) after landing on Runway 14
Airplane #2 is waiting for 45 microseconds before landing again
Airplane #1 is waiting for 42 microseconds before landing again
Airplane #5 is waiting for 31 microseconds before landing again
Airplane #7 is acquiring any needed runway(s) for landing on Runway 4R

Checking airport status for requested Runway 4R...
Number of simultaneous landing requests == 1, max == 4
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 #7 is taxiing on Runway 4R for 5 microseconds
Airplane #6 is acquiring any needed runway(s) for landing on Runway 4R
Airplane #7 is releasing any needed runway(s) after landing on Runway 4R

Checking airport status for requested Runway 4R...
Number of simultaneous landing requests == 2, max == 4
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 #6 is taxiing on Runway 4R for 1 microseconds
Airplane #7 is waiting for 35 microseconds before landing again
Airplane #6 is releasing any needed runway(s) after landing on Runway 4R
Airplane #6 is waiting for 59 microseconds before landing again
Airplane #3 is acquiring any needed runway(s) for landing on Runway 15L

Checking airport status for requested Runway 15L...
Number of simultaneous landing requests == 1, max == 4
Number of planes landing on runway 4L == 0
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 == 1
Number of planes landing on runway 15R == 0
Status check complete, no rule violations (yay!)
Airplane #3 is taxiing on Runway 15L for 8 microseconds
Airplane #3 is releasing any needed runway(s) after landing on Runway 15L
Airplane #3 is waiting for 83 microseconds before landing again
```

Airplane #1 is acquiring any needed runway(s) for landing on Runway 4R

Checking airport status for requested Runway 4R...
Number of simultaneous landing requests == 1, max == 4
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 #1 is taxiing on Runway 4R for 4 microseconds
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 == 2, max == 4
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 5 microseconds
Airplane #4 is acquiring any needed runway(s) for landing on Runway 4R
Airplane #5 is acquiring any needed runway(s) for landing on Runway 15R
Airplane #2 is releasing any needed runway(s) after landing on Runway 14
Airplane #1 is releasing any needed runway(s) after landing on Runway 4R
Airplane #7 is acquiring any needed runway(s) for landing on Runway 4R

Checking airport status for requested Runway 4R...
Number of simultaneous landing requests == 3, max == 4
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 microseconds
Airplane #4 is releasing any needed runway(s) after landing on Runway 4R

Checking airport status for requested Runway 15R...
Number of simultaneous landing requests == 4, max == 4
Number of planes landing on runway 4L == 0
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 == 1
Status check complete, no rule violations (yay!)
Airplane #5 is taxiing on Runway 15R for 5 microseconds
Airplane #2 is waiting for 50 microseconds before landing again
Airplane #1 is waiting for 51 microseconds before landing again
Airplane #6 is acquiring any needed runway(s) for landing on Runway 4R
Airplane #5 is releasing any needed runway(s) after landing on Runway 15R

Checking airport status for requested Runway 4R...
Number of simultaneous landing requests == 3, max == 4
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 #6 is taxiing on Runway 4R for 5 microseconds
Airplane #4 is waiting for 12 microseconds before landing again
Airplane #6 is releasing any needed runway(s) after landing on Runway 4R

Checking airport status for requested Runway 4R...
Number of simultaneous landing requests == 3, max == 4
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 #7 is taxiing on Runway 4R for 7 microseconds
Airplane #5 is waiting for 17 microseconds before landing again
Airplane #6 is waiting for 58 microseconds before landing again
Airplane #7 is releasing any needed runway(s) after landing on Runway 4R
Airplane #7 is waiting for 6 microseconds before landing again
Airplane #3 is acquiring any needed runway(s) for landing on Runway 15L

Checking airport status for requested Runway 15L...
Number of simultaneous landing requests == 1, max == 4
Number of planes landing on runway 4L == 0
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 == 1
Number of planes landing on runway 15R == 0
Status check complete, no rule violations (yay!)
Airplane #3 is taxiing on Runway 15L for 6 microseconds
Airplane #3 is releasing any needed runway(s) after landing on Runway 15L
Airplane #3 is waiting for 20 microseconds before landing again
Airplane #2 is acquiring any needed runway(s) for landing on Runway 9
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 == 4
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 #2 is taxiing on Runway 9 for 7 microseconds
Airplane #4 is acquiring any needed runway(s) for landing on Runway 4L
Airplane #2 is releasing any needed runway(s) after landing on Runway 9
Airplane #5 is acquiring any needed runway(s) for landing on Runway 4L
Airplane #7 is acquiring any needed runway(s) for landing on Runway 14
Airplane #6 is acquiring any needed runway(s) for landing on Runway 4R

Checking airport status for requested Runway 9...
Number of simultaneous landing requests == 2, max == 4
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 9 microseconds
Airplane #1 is releasing any needed runway(s) after landing on Runway 9

Checking airport status for requested Runway 4L...
Number of simultaneous landing requests == 3, max == 4
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 #4 is taxiing on Runway 4L for 1 microseconds
Airplane #2 is waiting for 84 microseconds before landing again
Airplane #4 is releasing any needed runway(s) after landing on Runway 4L

Checking airport status for requested Runway 4L...
Number of simultaneous landing requests == 3, max == 4
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 #5 is taxiing on Runway 4L for 5 microseconds
```

Makefile

```
CC = g++
CFLAGS = -c -g -Og -std=c++11
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)
```

```
// File: Airport.cpp
/*
 * Name: Ronney Sanchez
 * Date: November 27, 2018
 * Course: COMP2040 Computing 4
 * Assignment: Airport Simulation
*/

/**
*  Airport driver program
*/

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

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

using namespace std;

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([=]{ap->land();}));
	}

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

	return 0;

} // end main
```

```
/* Name: Ronney Sanchez
 * Date: November 27, 2018
 * Course: COMP2040 Computing 4
 * Assignment: Airport Simulation
*/

/**
*  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
```

```
/* File: Airplane.cpp
 * Name: Ronney Sanchez
 * Date: November 27, 2018
 * Course: COMP2040 Computing 4
 * Assignment: Airport Simulation
*/

#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
```

```
/*File: AirportRunways.hpp
 * Name: Ronney Sanchez
 * Date: November 27, 2018
 * Course: COMP2040 Computing 4
 * Assignment: Airport Simulation
*/

/**
*  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
```

```
/*File: AirportRunways.cpp
 * Name: Ronney Sanchez
 * Date: November 27, 2018
 * Course: COMP2040 Computing 4
 * Assignment: Airport Simulation
*/

#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()
```

```
/* File: AirportServer.hpp
 * Name: Ronney Sanchez
 * Date: November 27, 2018
 * Course: COMP2040 Computing 4
 * Assignment: Airport Simulation
*/

/**
*  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
	*/
	 // Used to enforce mutual exclusion for acquiring & releasing runways

	/**
	 *  ***** Add declarations of your own Locks and Condition Variables here *****
	 */
	mutex conditionLock;
	mutex runway4L;
	mutex runway4R;
	mutex runway9;
	mutex runway14;
	mutex runway15L;
	mutex runway15R;
	condition_variable condition;

}; // end class AirportServer

#endif
```

```
1   /* File: AirportServer.cpp
2    * Name: Ronney Sanchez
3    * Date: November 27, 2018
4    * Course: COMP2040 Computing 4
5    * Assignment: Airport Simulation
6   */
7
8   #include <iostream>
9   #include <thread>
10  #include <condition_variable>
11
12  #include "AirportServer.h"
13
14  //Comment out Lines 16, 32, 64, 80
15
16  /**
17  *  Called by an Airplane when it wishes to land on a runway
18  */
19  void AirportServer::reserveRunway(int airplaneNum, AirportRunways::RunwayNumber
20  runway)
21  {
22          // Acquire runway(s)
23          {  // Begin critical region
24                  //unique_lock<mutex> runwaysLock(runwaysMutex);
25
26                  {
27                          lock_guard<mutex> lk(AirportRunways::checkMutex);
28
29                          cout << "Airplane #" << airplaneNum << " is acquiring any needed
30  runway(s) for landing on Runway "
31                                  << AirportRunways::runwayName(runway) << endl;
32                  }
33                  unique_lock<mutex> condition_lock(conditionLock);
34                  /**
35                   *  ***** Add your synchronization here! *****
36                   */
37
38                  AirportRunways::incNumLandingRequests();
39
40                  while(AirportRunways::getNumLandingRequests() >
41  AirportRunways::MAX_LANDING_REQUESTS)
42                  {
43                          condition.wait(condition_lock);
44                  }
45
46                  switch(runway)
```

```
47                  {
48
49                          case AirportRunways::RunwayNumber::RUNWAY_4L:
50                                  runway4L.lock();
51                                  runway15L.lock();
52                                  runway15R.lock();
53                          break;
54
55                          case AirportRunways::RunwayNumber::RUNWAY_4R:
56                                  runway4R.lock();
57                                  runway9.lock();
58                                  runway15L.lock();
59                                  runway15R.lock();
60                          break;
61
62                          case AirportRunways::RunwayNumber::RUNWAY_9:
63                                  runway9.lock();
64                                  runway4R.lock();
65                                  runway15R.lock();
66                          break;
67
68                          case AirportRunways::RunwayNumber::RUNWAY_14:
69                                  runway14.lock();
70                          break;
71
72                          case AirportRunways::RunwayNumber::RUNWAY_15L:
73                                  runway15L.lock();
74                                  runway4L.lock();
75                                  runway4R.lock();
76                          break;
77
78                          default:
79                                  runway15R.lock();
80                                  runway4L.lock();
81                                  runway4R.lock();
82                          break;
83                  }
84
85                  AirportRunways::checkAirportStatus(runway);
86
87          } // End critical region
88
89          // obtain a seed from the system clock:
90          unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
91          std::default_random_engine generator(seed);
92
```

```
	// 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! *****
		*/

		AirportRunways::finishedWithRunway(runway);

	    switch(runway)
		{
			case AirportRunways::RunwayNumber::RUNWAY_4L:
			runway4L.unlock();
```

```
                runway15L.unlock();
                runway15R.unlock();
            break;

                case AirportRunways::RunwayNumber::RUNWAY_4R:
                runway4R.unlock();
                runway9.unlock();
                runway15L.unlock();
                runway15R.unlock();
                break;

                case AirportRunways::RunwayNumber::RUNWAY_9:
                runway9.unlock();
                runway4R.unlock();
                runway15R.unlock();
                break;

                case AirportRunways::RunwayNumber::RUNWAY_14:
                runway14.unlock();
                break;

                case AirportRunways::RunwayNumber::RUNWAY_15L:
                runway15L.unlock();
                runway4L.unlock();
                runway4R.unlock();
            break;

                default:
                runway15R.unlock();
                runway4L.unlock();
                runway4R.unlock();
                break;
        }
            unique_lock<mutex> condition_lock(conditionLock);
            // Update the status of the airport to indicate that the landing is complete
            AirportRunways::decNumLandingRequests();
            //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()
```