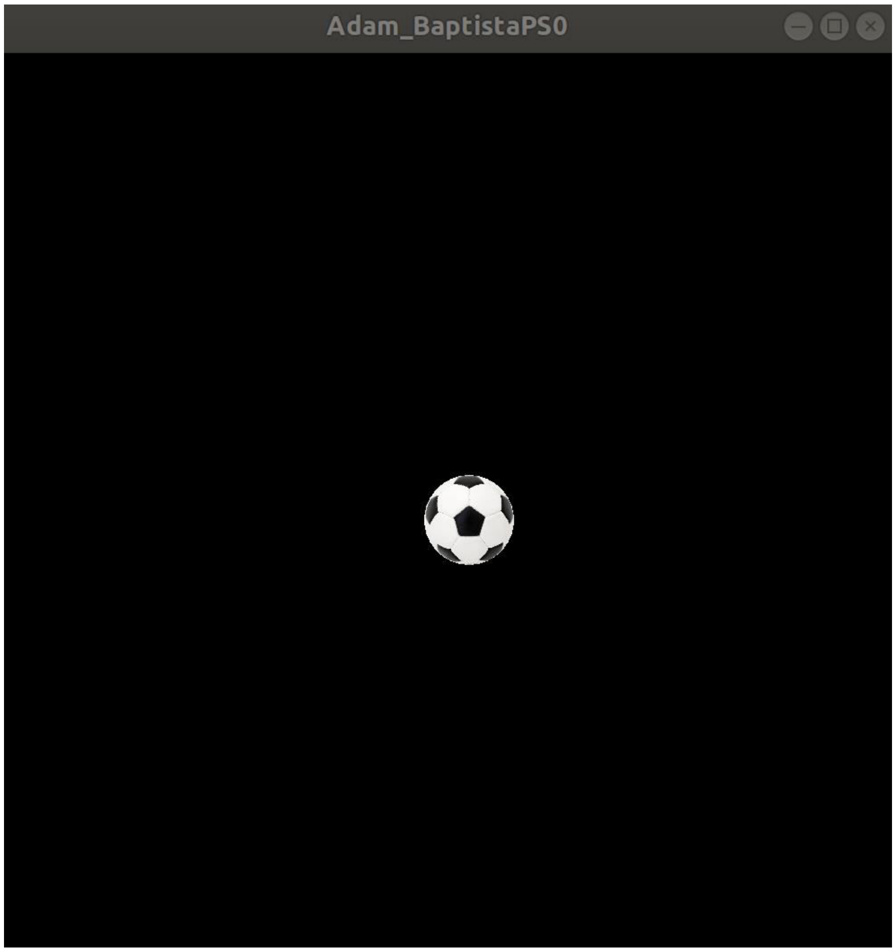
Adam Baptista
COMP IV: Project Portfolio
Fall 2019

<u>Contents:</u>

PS0 Hello World Assignment

The purpose of the first assignment was to get us used to using the SFML library to display different objects and take in key strokes to make an object move on the screen. In my project, I implemented 3 different functionalities. The first functionality that I had added was that the esc button would close the program. The second functionality that I had added was that the space button would move the sprite back to the middle of the window. The final thing that I added to the project was that the arrow keys would move the object in their respective direction, and each direction had a different color based on which arrow key was pressed.

In this project I had learned how to use the SFML library to create a window, sprites, load images to sprites, take in key strokes, and finally add shapes and change the color of shapes.

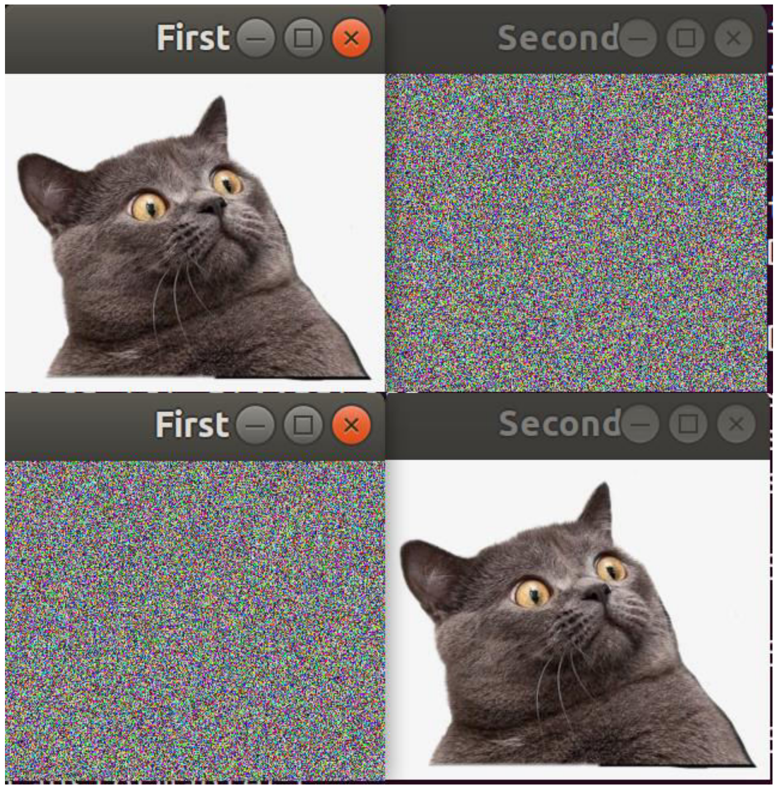


```cpp
//Adam Baptista

#include <SFML/Graphics.hpp>

int main()
{
        sf::RenderWindow window(sf::VideoMode(1000, 1000), "Adam_BaptistaPS0");
        sf::CircleShape shape(50.f);
        shape.setFillColor(sf::Color::Red);

        sf::Texture texture;
        if (!texture.loadFromFile("sprite.jpg"))
                return EXIT_FAILURE;
        shape.setTexture(&texture);
        shape.setPosition(500, 500);
        shape.setOrigin(25, 25);


        while (window.isOpen())
        {
                sf::Event event;
                while (window.pollEvent(event))
                {
                        if (event.type == sf::Event::Closed)
                                window.close();
                        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
                        {
                                shape.setFillColor(sf::Color::Blue);
                                shape.move(-5, 0);
                        }
                        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
                        {
                                shape.setFillColor(sf::Color::Green);
                                shape.move(5, 0);
                        }
                        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
                        {
                                shape.setFillColor(sf::Color::Red);
                                shape.move(0, -5);
                        }
                        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
                        {
                                shape.setFillColor(sf::Color::Yellow);
                                shape.move(0, 5);
                        }
                        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))
                                shape.setPosition(500, 500);
                        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
                                window.close();
                        else
                                shape.setFillColor(sf::Color::White);

                }

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

        return 0;
}
```

PS1 Linear Feedback Shift Register

In this assignment, I had to encrypt an image using the LFSR generate command and save that image, then decrypt that image to get the original image using the same LFSR generate command. OO designs that were central to this assignment were classes. An additional implementation that I added to this assignment was that the esc key would close the window.

There was only serious problem I encountered was that I copied and pasted p.r ^ lfsr.generate(5) for the p.g and p.b and it took me forever to find out what was wrong with it, and the make file.



```
CC = g++
CFLAGS = -std=c++11 -c -g -Wall -Werror -pedantic
LIBS = -lsfml-graphics -lsfml-window -lsfml-system

all: PhotoMagic

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

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

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

clean:
	rm *.o PhotoMagic
```

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

LFSR::LFSR(string seed, int tap) {
        for (unsigned int i = 0; i < seed.length(); i++)
                this->seed.push_back(seed[i]);
        //save_seed = seed;
        //since the tap is counted from right to left, must take total lengt
h
        //and subtract it from the input tap
        this->tap = seed.length() - tap - 1;
}

int LFSR::step() {
        int first = seed.at(0),
            _tap = seed.at(tap),
                n_bit = first ^ _tap;
        seed.erase(seed.begin());
        seed.push_back(n_bit);
        return n_bit;
}

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

ostream& operator<< (ostream &out, const LFSR &obj) {
        for (unsigned int i = 0; i < obj.seed.size(); i++) {
                out << obj.seed[i];
        }

        return out;
}

//LFSR::~LFSR() {}
```

```
#include <iostream>
#include <string.h>
#include <vector>
using namespace std;

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

        int step();

        int generate(int k);

        friend ostream& operator<< (ostream &out, const LFSR &obj);

        //~LFSR();

private:
        vector<int> seed;
        int tap;
};
```

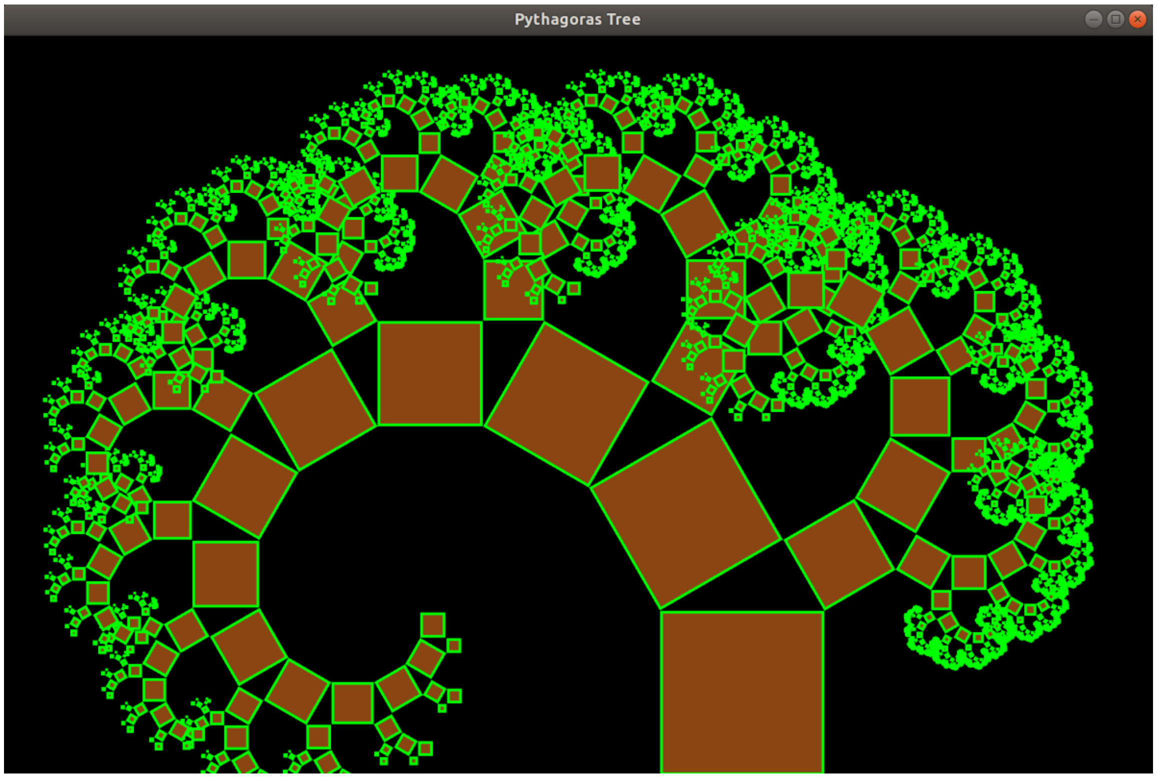
```
// pixels.cpp:
// using SFML to load a file, manipulate its pixels, write it to disk
// Fred Martin, fredm@cs.uml.edu, Sun Mar  2 15:57:08 2014

// g++ -o pixels pixels.cpp -lsfml-graphics -lsfml-window

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


int main(int argc, char* argv[])
{
        string input_file = argv[1];
        string output_file = argv[2];
        string seed = argv[3];
        int tap = atoi(argv[4]);

        sf::Image first;
        if (!first.loadFromFile(input_file))
                return -1;

        sf::Image second;
        if (!second.loadFromFile(input_file))
                return -1;

        // p is a pixel
        sf::Color p;
        sf::Vector2u win1_size = first.getSize();

        LFSR lfsr(seed, tap);

        // create encrypted image of the original image
        for (unsigned int x = 0; x < win1_size.x; x++) {
                for (unsigned int y = 0; y < win1_size.y; y++) {
                        p = second.getPixel(x, y);
                        p.r = p.r ^ lfsr.generate(5);
                        p.g = p.g ^ lfsr.generate(5);
                        p.b = p.b ^ lfsr.generate(5);
                        second.setPixel(x, y, p);
                }
        }

        sf::RenderWindow window1(sf::VideoMode(win1_size.x, win1_size.y), "F
irst");
        sf::RenderWindow window2(sf::VideoMode(win1_size.x, win1_size.y), "S
econd");

        sf::Texture original;
        original.loadFromImage(first);
        sf::Texture encrypted;
        encrypted.loadFromImage(second);

        sf::Sprite sprite1;
        sprite1.setTexture(original);
        sf::Sprite sprite2;
        sprite2.setTexture(encrypted);

        while (window1.isOpen() && window2.isOpen()) {
        sf::Event event;
```

```
                while (window1.pollEvent(event)) {
                        if (event.type == sf::Event::Closed)
                                window1.close();
                        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
                                window1.close();
                }
                while (window2.pollEvent(event)) {
                        if (event.type == sf::Event::Closed)
                                window2.close();
                        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
                                window2.close();
                        }
                window1.clear();
                window1.draw(sprite1);
                window1.display();
                window2.clear();
                window2.draw(sprite2);
                window2.display();
        }

        // fredm: saving a PNG segfaults for me, though it does properly
        //   write the file
        if (!second.saveToFile(output_file))
                return -1;

        return 0;
}
```

PS2 Recursive Graphics (Pythagoras tree)

In this assignment, I coded the Pythagoras tree using the SFML library and recursion. A key algorithm I used to design the tree was to set the origin of the left square to the bottom left and the position of the origin to the top left of the previously drawn square, then rotated +45 deg. The right side was similar which was that I set the origin of the right square to the bottom right and the positon to the top right of the previously drawn square, then rotated -45 deg. Anohter OO design that was central to this assignment were classes. I also implemented an additional functionality where the user could use the right arrow key to step through each recursive step of drawing the tree.

In this assignment I learned how to use recursion to draw recursive objects. The majority of the time I spent on this assignment was figuring out the recursuve part of the assignment because I had a hard time setting the different origins and positions correctly. I also didn't include the drawable class because I do not understand how to implent it, but I just got the adress of the window, so I could draw the shape within the function, which worked fine.



```
CC = g++
CFLAGS = -std=c++11 -c -g -Wall -Werror -pedantic
LIBS = -lsfml-graphics -lsfml-window -lsfml-system

all: tree

tree: main.o PTree.o
	$(CC) main.o PTree.o -o tree $(LIBS)

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

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

clean:
	rm *.o *.gch tree
```

```
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <cmath>
#include <iostream>
#include <time.h>
#include <string>

using namespace sf;

class PTree{
public:
    PTree();

    void pTree(RenderWindow &target, int size, Vector2f pos, Vector2f orig, int deg, int iter);

private:
    ConvexShape shape;
};
```

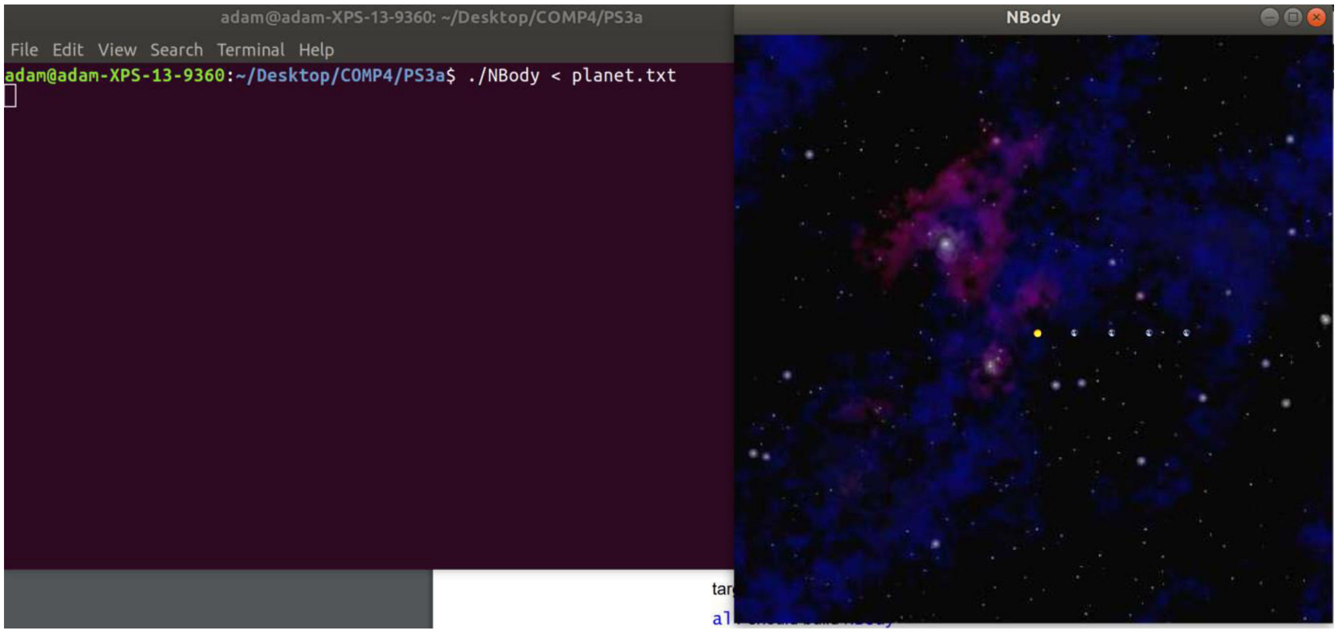
```cpp
#include "PTree.hpp"

#define RT sqrt(2)

PTree::PTree() {}

void PTree::pTree(RenderWindow &target, int size, Vector2f pos, Vector2f ori
g, int deg, int iter) {
    Vector2f Lp, Rp, origL(0, (size/2)*sqrt(3)), origR(size/2, size/2);
    Color Brown(139, 69, 19);

    shape.setPointCount(4);
    shape.setPoint(0, Vector2f(0, 0));
    shape.setPoint(1, Vector2f(0, size));
    shape.setPoint(2, Vector2f(size, size));
    shape.setPoint(3, Vector2f(size, 0));

    shape.setPosition(pos);
    shape.setOrigin(orig);
    shape.setRotation(deg);

    Rp = shape.getTransform().transformPoint(shape.getPoint(0));
    Lp = shape.getTransform().transformPoint(shape.getPoint(3));

    shape.setOutlineColor(Color::Green);
    shape.setFillColor(Brown);
    shape.setOutlineThickness(-5);
    if (iter < 0)
        return;
    Rp = shape.getTransform().transformPoint(shape.getPoint(0));
    Lp = shape.getTransform().transformPoint(shape.getPoint(3));

    target.draw(shape);
    //pTree(target, (size/2)*sqrt(3), Rp, origL, deg-30, iter-1);
    //pTree(target, size / 2, Lp, origR, deg+60, iter-1);

    pTree(target, (size/2)*sqrt(3), Rp, origL, deg-30, iter-1);
    pTree(target, size/2, Lp, origR, deg+60, iter-1);
}
```

```
#include "PTree.hpp"

int main(int argc, char* argv[])
{
    int L, N, iter = 0;
    L = atoi(argv[1]);
    N = atoi(argv[2]);

    std::cout << "Use right arrow to go to the next iteration." << std::endl
;

    RenderWindow window (VideoMode(7*L, 4.5*L), "Pythagoras Tree");

    //Vector2f pos(6*L/2-L/2, 4*L), orig(0, L);
    Vector2f pos(6*L/1.5, 4.5*L), orig(0, L);

    PTree rD;

    while(window.isOpen())
    {
        Event event;
        while(window.pollEvent(event))
        {
            if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboa
rd::Escape))
                window.close();
            else if(Keyboard::isKeyPressed(Keyboard::Right)) {
                if (iter < N)
                    iter++;
            }
        }

        window.clear();
        rD.pTree(window, L, pos, orig, 0, iter);
        window.display();
    }

    return EXIT_SUCCESS;
}
```

PS3 N-Body Simulation

In this project, I used physics and the SFML library to simulate celestial bodies and gravity graphically. One key OO design that was central to this assignment was the use of classes to create different bodies. In addition to that, I also used unique pointers to be able to draw multiple different sprites for each different celestial body in the simulation. To print the state of the universe at the end of the simulation I just outputted the current x and y positions, and the x and y velocities at the time the window is closed. To make the planets rotate counter clockwise, I just -= for all acceleration and velocities, because when it was +=, it was going clockwise. Used shared pointers from before. To play the song, I had to change the makefile to include -lsfml-audio in the LIBS, otherwise it didnt work. To display text I had to download a .ttf file to get text, and I also had to look online on how to do it. I also added music and a timer to the program as well.

One difficulty was that it took me forever to figure out how to make the planets move reasonably, and I was stuck the majority of the time with the planets just going up, or just going in one direction, or disappearing.


adam@adam-XPS-13-9360: ~/Desktop/COMP4/PS3a
File Edit View Search Terminal Help
NBody

```
adam@adam-XPS-13-9360:~/Desktop/COMP4/PS3a$ ./NBody < planet.txt
```

```
CC = g++
CFLAGS = -std=c++11 -c -g -Wall -Werror -pedantic
LIBS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio

all: NBody

NBody: main.o planets.o
	$(CC) main.o -o NBody $(LIBS)

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

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

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

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

#include <iostream>
#include <vector>
#include <string>
#include <memory>
#include <math.h>

#define G 6.67e-11

using namespace std;
using namespace sf;

class Body : public Drawable {
private:
    Texture texture;
    Vector2u winSize;

public:
    Sprite sprite;
    Vector2f F;
    string img_file;
    double R, x ,y, xVel, yVel, mass;

    Body();

    Vector2f getPos();

    double getM();

    void scale(Vector2u winSize, double R);

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

    friend istream &operator>>(istream &in, Body &body);

    void time(double time);

    void move();

    void setV(double ax, double ay, double time);

    void setPos(double time);

    ~Body();
};

double getRadius(double body1_pos, double body2_pos);

double getForce(double mass1, double mass2, double r);

double dirF(double F, double dF, double r);



```

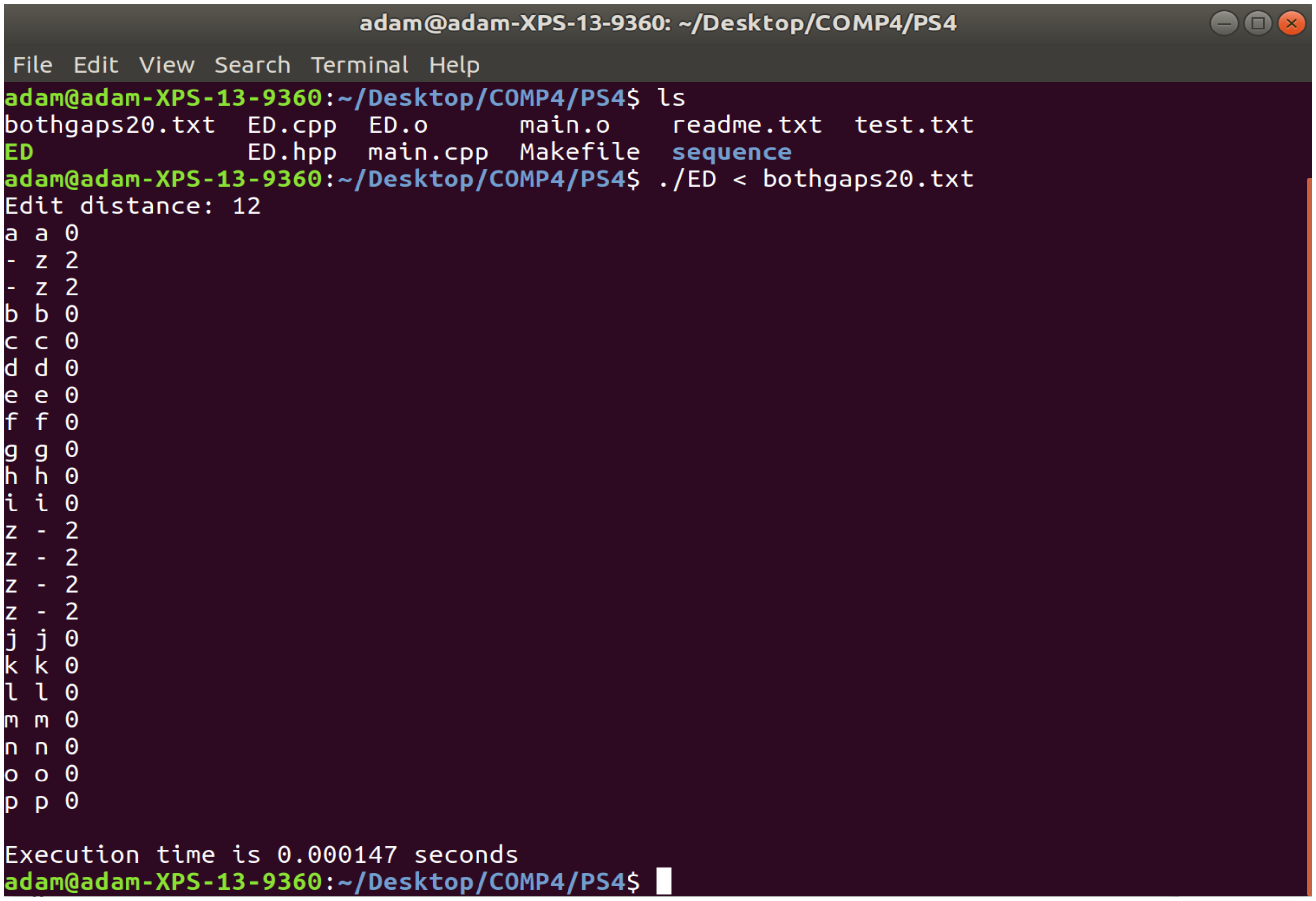
```cpp
#include "planets.hpp"

Body::Body() {}

Vector2f Body::getPos() {
    Vector2f pos(xVel, yVel);
    return pos;
}

double Body::getM() {
    return mass;
}

void Body::scale(Vector2u winSize, double R) {
    this->winSize = winSize;
    this->R = R;
}

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

istream &operator >>(istream &in, Body &body) {
    in >> body.x >> body.y >> body.xVel >> body.yVel
        >> body.mass >> body.img_file;

    if (!body.texture.loadFromFile("nbody/" + body.img_file)) {
        cout << "Failed to load image " << body.img_file << endl;
        exit(EXIT_FAILURE);
    }
    body.sprite.setTexture(body.texture);
    return in;
}

void Body::time(double time) {
    double ax = F.x / mass;
    double ay = F.y / mass;

    setV(ax, ay, time);

    setPos(time);
}

void Body::setV(double ax, double ay, double time) {
    xVel -= (ax * time);
    yVel -= (ay * time);
}

void Body::setPos(double time) {
    x -= xVel * time;
    y -= yVel * time;
}

void Body::move() {
    double newX = ((x / R) * winSize.x/2) + winSize.x / 2;
    double newY = ((y / R) * winSize.y/2) + winSize.y / 2;
    Vector2f middle;
    middle.x = sprite.getTexture()->getSize().x / 2;
    middle.y = sprite.getTexture()->getSize().y / 2;
    sprite.setOrigin(middle);

    sprite.setPosition(newX, newY);
}

double getRadius(double body1_pos, double body2_pos) {
    return sqrt(pow(body1_pos, 2) + pow(body2_pos, 2));
}

double getForce(double mass1, double mass2, double r) {

    return (G * mass1 * mass2) / pow(r, 2);;
}

double dirF(double F, double dF, double r) {
    double fDir = F * dF / r;
    return fDir;
}

Body::~Body() {}
```

```
#include <sstream>

#include "planets.cpp"

#define BACKGROUND "background.jpg"

Vector2u WINSIZE, backSize;

int main(int argc, char* argv[]) {
    int N;
    double R;
    double T = atoi(argv[1]);
    double deltaT = atoi(argv[2]);
    vector<shared_ptr<Body>> bodies;
    double dx = 0,
           dy = 0,
           r = 0,
           f = 0,
           fX = 0,
           fY = 0;

    RenderWindow window(VideoMode(800, 800), "NBody");

    Sprite background;
    Texture texture;
    if (!texture.loadFromFile(BACKGROUND)) {
        cout << "Failed to load background" << endl;
        return EXIT_FAILURE;
    }

    Music music;
    if (!music.openFromFile("HEYYEYAAEYAAAEYAEYAA.ogg")) {
        cout << "Failed to load music" << endl;
        return EXIT_FAILURE;
    }
    music.play();

    Font font;
    font.loadFromFile("digital-7 (mono).ttf");
    Text text;
    text.setFont(font);
    text.setPosition(0, 0);
    text.setCharacterSize(24);
    stringstream timer;

    backSize = texture.getSize();
    WINSIZE = window.getSize(); //gets the window size

    double xScale = (double) WINSIZE.x / backSize.x;
    double yScale = (double) WINSIZE.y / backSize.y;

    background.setTexture(texture);
    background.setScale(xScale, yScale);

    window.setFramerateLimit(60);

    cin >> N;
    cin >> R;
    for (int i = 0; i < N; i++) {
        shared_ptr<Body> ptrBody(new Body());
        cin >> *ptrBody;
        ptrBody->scale(WINSIZE, R);
        bodies.push_back(ptrBody);
    }

    Clock clock;
    clock.restart();

    while (window.isOpen())
    {
        Time ElapsedTime = clock.getElapsedTime();
        double timePassed = ElapsedTime.asSeconds();
        timer.str(string());
        timePassed *= deltaT;
        timer << "Time passed in seconds: " << timePassed;
        text.setString(timer.str().c_str());
        Event event;
        while (window.pollEvent(event))
        {
            if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboard::Escape))
                window.close();
        }

        window.clear();
        window.draw(background);
        window.draw(text);

        for (unsigned int i = 0; i < bodies.size(); i++) {
            fX = 0;
            fY = 0;
            for (unsigned int j = 0; j < bodies.size(); j++) {
                if (i != j) {
                    dx = bodies[j]->x - bodies[i]->x;
                    dy = bodies[j]->y - bodies[i]->y;
                    r = getRadius(dx, dy);
                    f = getForce(bodies[i]->getM(), bodies[j]->getM(), r);
                    fX += dirF(dx, f, r);
                    fY += dirF(dy, f, r);
                }
            }
            bodies[i]->F.x = fX;
            bodies[i]->F.y = fY;
            bodies[i]->time(deltaT);
            bodies[i]->move();
            window.draw(*bodies[i]);
        }

        if (timePassed > T || Keyboard::isKeyPressed(Keyboard::Escape) || event.type == Event::Closed) {
            for (unsigned int i = 0; i < bodies.size(); i++) {
                cout << " " << bodies[i]->x;
                cout << " " << bodies[i]->y;
                cout << " " << bodies[i]->xVel;
                cout << " " << bodies[i]->yVel;
                cout << " " << bodies[i]->mass;
                cout << " " + bodies[i]->img_file << endl;
            }
        }

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

PS4 DNA Sequence Alignment

In this project, I had to use the Needleman-Wunsch method to find the best allignment for two strings. In this assignment OO designs that were implemented were classes, and data structures used were arrays to create a 2d array in order to implement the Needlema-Wunsch method. In this assignment I learned different ways to use valgrind in order to find memory leaks, and the SFML library in order to determine the run time for different sequences.

One problem that I ran into was that I got stuck on how to delete a 2d array to get rid of memory leaks, but to solve this problem I used google to help me with it.


adam@adam-XPS-13-9360: ~/Desktop/COMP4/PS4
File Edit View Search Terminal Help

```
adam@adam-XPS-13-9360:~/Desktop/COMP4/PS4$ ls
bothgaps20.txt  ED.cpp  ED.o      main.o    readme.txt  test.txt
ED              ED.hpp  main.cpp  Makefile  sequence
adam@adam-XPS-13-9360:~/Desktop/COMP4/PS4$ ./ED < bothgaps20.txt
Edit distance: 12
a a 0
- z 2
- z 2
b b 0
c c 0
d d 0
e e 0
f f 0
g g 0
h h 0
i i 0
z - 2
z - 2
z - 2
z - 2
j j 0
k k 0
l l 0
m m 0
n n 0
o o 0
p p 0

Execution time is 0.000147 seconds
adam@adam-XPS-13-9360:~/Desktop/COMP4/PS4$ 
```

```
CC = g++
CFLAGS = -std=c++11 -c -g -Wall -Werror -pedantic
LIBS = -lsfml-system

all: ED

ED: main.o ED.o
	$(CC) main.o -o ED $(LIBS)

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

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

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

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

using namespace std;
using namespace sf;

class ED {
public:
    ED(string a, string b);

    int penalty(char a, char b);
    int min(int a, int b, int c);
    int OptDistance();
    string Alignment();
    int getCost();

    ~ED();

private:
    string x, y;
    int M, N, cost;
    int** opt;

};
```

```
#include "ED.hpp"

ED::ED(string a, string b) {

    this->x = a;
    this->y = b;

    x += '-';
    y += '-';

    M = x.length() + 1;
    N = y.length() + 1;

    opt = new int*[M];
    for (int i = 0; i < M; i++)
        opt[i] = new int[N];
}

int ED::penalty(char a, char b) {
    if (a == b)
        return 0;
    else
        return 1;
}

int ED::min(int a, int b, int c) {
    if (a < b && a < c)
        return a;
    else if (b < a && b < c)
        return b;
    else
        return c;
}
int ED::OptDistance() {

    //add bottom outside
    int j = 0;
    for (int i = N-1; i >= 0; i--) {
        opt[M-1][i] = j;
        j += 2;
    }

    //add right outisde
    j = 0;
    for (int i = M-1; i >= 0; i--) {
        opt[i][N-1] = j;
        j += 2;
    }

    //compare
    for (int i = M-2; i >= 0; i--) {
        for (int j = N-2; j >= 0; j--) {
            if (x[i] != y[j]) {
                opt[i][j] = min(opt[i+1][j] + 2, opt[i][j+1] + 2, opt[i+1][j
+1] + 1);
            }
            else
                opt[i][j] = opt[i+1][j+1];
        }
    }

    string out = Alignment();
    cout << "Edit distance: " << cost << endl;
    cout << out << endl;

    for (int i = 0; i < M; i++)
        delete [] opt[i];

    delete [] opt;
    return 0;
}
string ED::Alignment() {
    vector<char> out;

    int i = 0,
        j = 0,
        pen;;
    while (i < M-2|| j < N-2) {
        if (x[i] == y[j]) {
            pen = penalty(x[i], y[j]);
            out.push_back(x[i]);
            out.push_back(' ');
            out.push_back(y[j]);
            out.push_back(' ');
            i++;
            j++;
        }
        else if (opt[i][j] == opt[i+1][j+1] + 1) {
            pen = penalty(x[i], y[j]);
            out.push_back(x[i]);
            out.push_back(' ');
            out.push_back(y[j]);
            out.push_back(' ');
            i++;
            j++;
        }
        else if (opt[i][j] == opt[i+1][j] + 2) {
            pen = penalty(x[i], y[j]) + 1;
            out.push_back(x[i]);
            out.push_back(' ');
            out.push_back('-');
            out.push_back(' ');
            i++;
        }
        else if (opt[i][j] == opt[i][j+1] + 2) {
            pen = penalty(x[i], y[j]) + 1;
            out.push_back('-');
            out.push_back(' ');
            out.push_back(y[j]);
            out.push_back(' ');
            j++;
        }
        out.push_back('0' + pen);
        cost += pen;
        out.push_back('\n');
    }
    string new_out(out.begin(), out.end());
    return new_out;
}

int ED::getCost() {
    return cost;
```

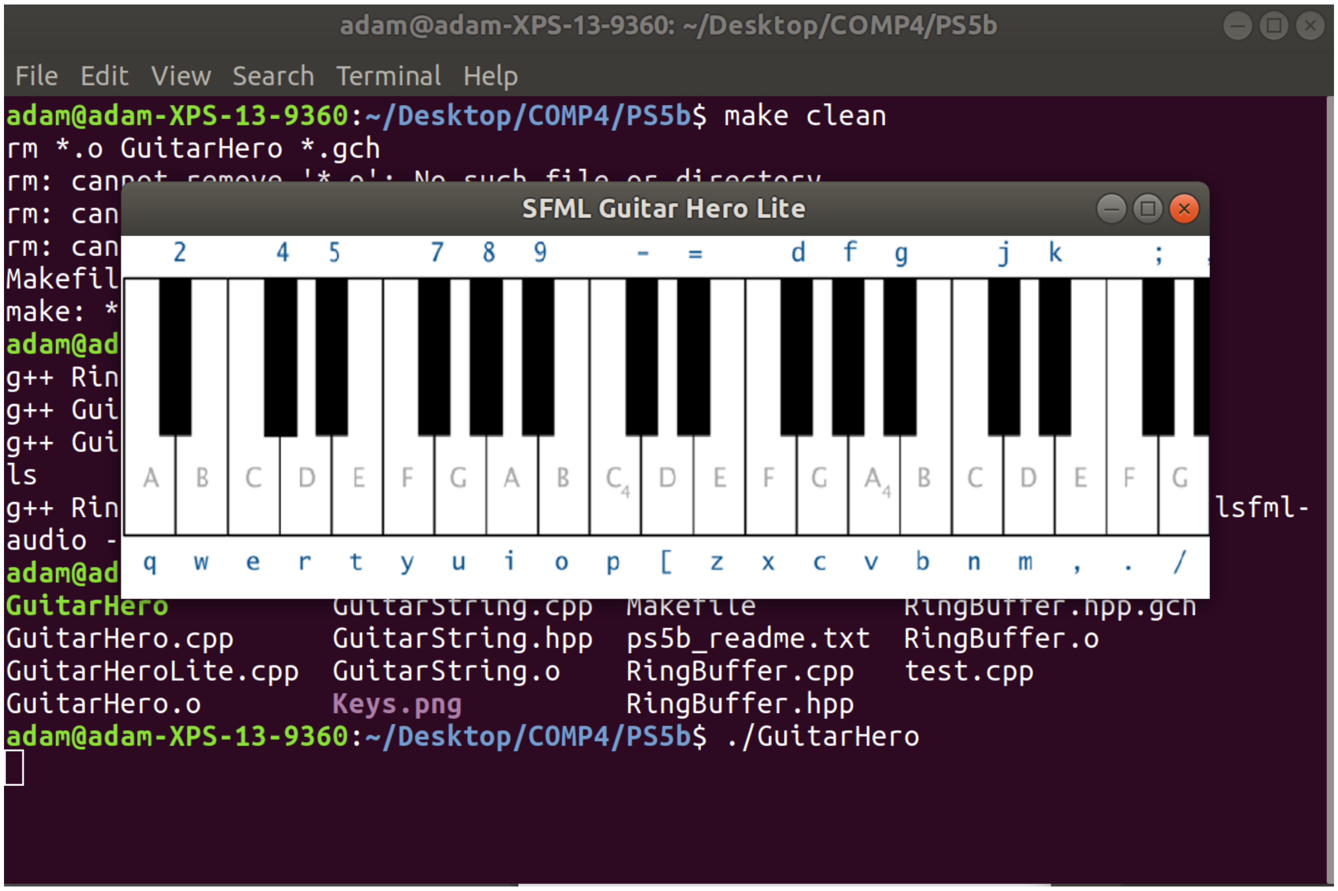
```
}

ED::~ED() {}
```

```
#include "ED.cpp"

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

    string a, b;
    cin >> a;
    cin >> b;

    ED ed(a, b);

    Clock clock;
    Time t;
    ed.OptDistance();
    t = clock.getElapsedTime();
    cout << "Execution time is " << t.asSeconds() << " seconds\n";
    //cout << "Edit distance (for longer sequences): " << ed.getCost() << endl;

    return 0;
}
```

PS5 Guitar Hero

In this assignment, I had to use the SFML library and a ring buffer to play different sounds when different keys were pressed. OO methods that were used in this assignment were classes, and data structures that were central to the assignment were vectors. Another key algorithm that was central to this assignment was queues. In this assignment I created a stack for queue, then implemented enqueue, dequeue, isfull, isempty, and peek for the queue stack. I also implemented a few other functions to help me debug my code while I was writing it such as size, print, and get.

One issue I had with this assignment was that only two of the keys on the keyboard that were supposed to create sounds did not create any sound.


adam@adam-XPS-13-9360: ~/Desktop/COMP4/PS5b
File Edit View Search Terminal Help
SFML Guitar Hero Lite
2 4 5 7 8 9 - = d f g j k ;
A B C D E F G A B C4 D E F G A4 B C D E F G
q w e r t y u i o p [z x c v b n m , . /

```
adam@adam-XPS-13-9360:~/Desktop/COMP4/PS5b$ make clean
rm *.o GuitarHero *.gch
rm: cannot remove '*.o': No such file or directory
rm: can
rm: can
Makefil
make: *
adam@ad
g++ Rin
g++ Gui
g++ Gui
ls
g++ Rin                                                                  lsfml-
audio -
adam@ad
GuitarHero          GuitarString.cpp   Makefile         RingBuffer.hpp.gch
GuitarHero.cpp      GuitarString.hpp   ps5b_readme.txt  RingBuffer.o
GuitarHeroLite.cpp  GuitarString.o     RingBuffer.cpp   test.cpp
GuitarHero.o        Keys.png           RingBuffer.hpp
adam@adam-XPS-13-9360:~/Desktop/COMP4/PS5b$ ./GuitarHero
```

```
CC = g++
CFLAGS = -std=c++11 -c -g -Wall -Werror# -pedantic
LIBS = -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window

all: GuitarHero

GuitarHero: RingBuffer.o GuitarString.o GuitarHero.o
	$(CC) RingBuffer.o GuitarString.o GuitarHero.o -o GuitarHero $(LIBS)

GuitarHero.o: RingBuffer.hpp GuitarString.hpp GuitarHero.cpp
	$(CC) GuitarHero.cpp $(CFLAGS)

GuitarString.o: RingBuffer.hpp GuitarString.hpp GuitarString.cpp
	$(CC) GuitarString.cpp $(CFLAGS)

RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
	$(CC) RingBuffer.cpp RingBuffer.hpp $(CFLAGS)

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

```
#include <SFML/Audio.hpp>
#include <SFML/System.hpp>

#include <vector>
#include <cstdlib>
#include <cmath>

#include "RingBuffer.hpp"

using namespace std;
using namespace sf;

class GuitarString {
public:
    GuitarString(double frequency);
    GuitarString(vector<Int16> init);

    void pluck();

    void tic();
    Int16 sample();
    int time();

private:
    RingBuffer rb;
    int count;

};
```

```
#include "GuitarString.hpp"

#define DECAY 0.996

GuitarString::GuitarString(double frequency) : rb(ceil(44100/frequency)) {
    count = 0;
}
GuitarString::GuitarString(vector<Int16> init) : rb(init.size()) {
    for (unsigned i = 0; i < init.size(); i++)
        rb.enqueue(init[i]);
    count = 0;
}

void GuitarString::pluck() {
    rb.empty();
    while (!rb.isFull())
        rb.enqueue((int16_t)(rand() & 0xffff));
}

void GuitarString::tic() {
    int N1 = rb.dequeue();
    int N2 = rb.peek();
    rb.enqueue(DECAY * 0.5 * (N1 + N2));
    count++;
}
Int16 GuitarString::sample() {
    return rb.peek();
}
int GuitarString::time() {
    return count;
}
```

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

#define DECAY 0.996

using namespace std;

class RingBuffer {
public:
    RingBuffer(int capacity);

    int ringSize();
    bool isEmpty();
    bool isFull();
    void enqueue(int16_t x);
    int16_t dequeue();
    int16_t peek();
    int get(int x);
    //void print();

    void empty();

private:
    int capacity, size, first, last;
    int16_t temp_first;
    std::vector<int16_t> vector;

};
```

```cpp
/*
Copyright 2019 Adam Baptista


*/

#include "RingBuffer.hpp"

RingBuffer::RingBuffer(int capacity) {
    try {
        if (capacity < 1)
            throw invalid_argument
                ("Capacity cannot be less than or equal to 1.");
    } catch(invalid_argument& e) {
        cerr << "RB constuctor: capacity cannot be less than or equal to 1.";
        cerr << endl;
        throw e;
    }
    //sets a certain amount of space for items, like vector = new int[capacity];
    vector.reserve(capacity);
    for (int i = 0; i < capacity; i++)
        vector.push_back(0);
    size = 0;
    first = 0;
    last = 0;
    this->capacity = capacity;
}

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

bool RingBuffer::isEmpty() {
    if (size > 0)
        return false;
    return true;
}

bool RingBuffer::isFull() {
    if (capacity > size)
        return false;
    return true;
}

void RingBuffer::enqueue(int16_t x) {
    try {
        if (isFull())
            throw runtime_error
                ("Enqueue: can’t enqueue an full ring");
    } catch (runtime_error& e) {
        cerr << "Enqueue: can’t enqueue an full ring";
        cerr << endl;
        throw e;
    }
    vector[last] = x;
    if (last == capacity - 1) {
        last = 0;
    } else {
        last++;
    }
    size++;
}

int16_t RingBuffer::dequeue() {
    try {
        if (isEmpty())
            throw runtime_error
                ("Dequeue: can't dequeue en empty buffer");
    } catch (runtime_error& e) {
        cerr << "Dequeue: can't dequeue en empty buffer";
        cerr << endl;
        throw e;
    }
    temp_first = peek();
    if (first == capacity - 1) {
        first = 0;
    } else {
        first++;
    }
    size--;
    return temp_first;
}

int16_t RingBuffer::peek() {
    try {
        if (isEmpty())
            throw runtime_error
                ("Peek: can't peek at an empty ring");
    } catch (runtime_error& e) {
        cerr << "Peek: can't peek at an empty ring";
        cerr << endl;
        throw e;
    }
    return vector[first];
}

/*
void RingBuffer::print() {
    for (int i = 0; i < capacity; i++)
        cout << vector[i] << endl;
    cout << endl;
}
*/

void RingBuffer::empty() {
    size = 0;
    first = 0;
    last = 0;
}
```

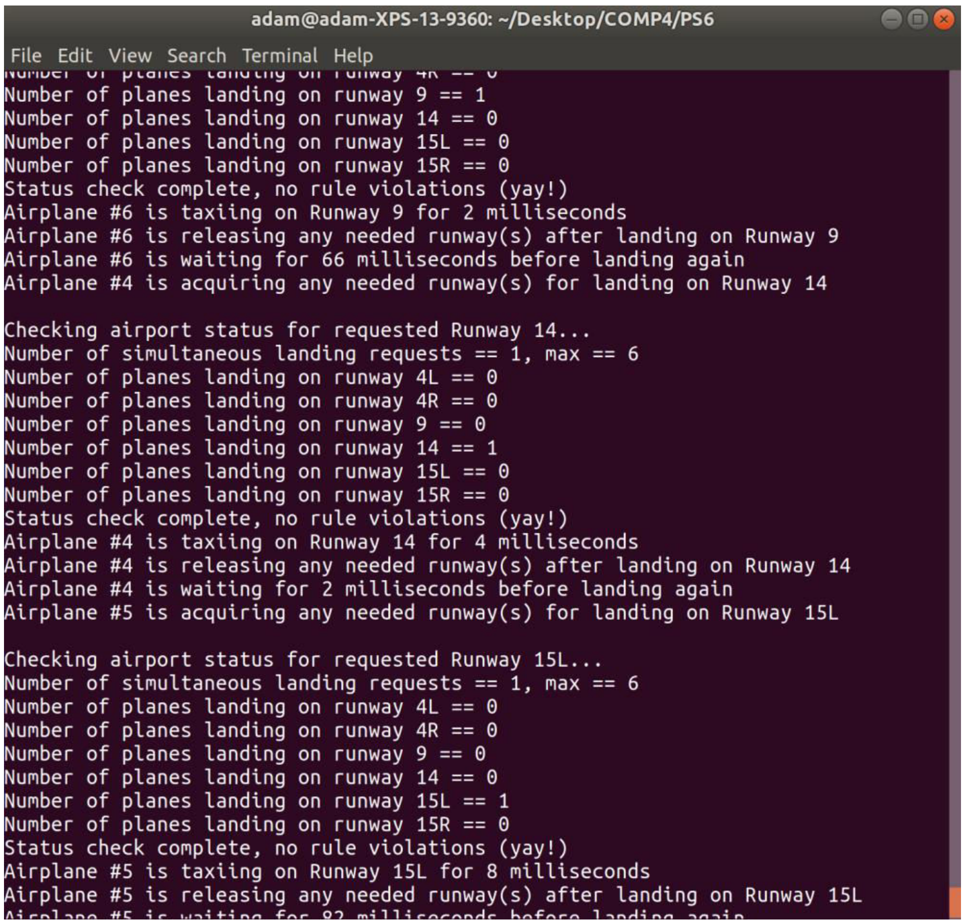
```
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>

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

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

#include "GuitarString.hpp"

#define HZ 44100

vector<int16_t> makeSampleFromString(GuitarString gs) {
vector<int16_t> samples;

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

int main() {
    Sprite background;
    Texture texture;
    if (!texture.loadFromFile("Keys.png")) {
        cout << "Failed to load background" << endl;
        return EXIT_FAILURE;
    }
    Vector2f backSize(texture.getSize());
    RenderWindow window(VideoMode(1200, 400), "SFML Guitar Hero Lite");
    background.setTexture(texture);

    Vector2f WINSIZE(window.getSize());
    double xScale = (double) WINSIZE.x / backSize.x;
    double yScale = (double) WINSIZE.y / backSize.y;

    background.setScale(xScale, yScale);


    Event event;
    double frequency;
    string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/â\200\231 ";
    vector<vector<int16_t> > samples(37);
    vector<Sound> sounds(37);
    vector<SoundBuffer> soundBuffers(37);

    for(int i = 0; i < 37; i++) {
        frequency = 440 * pow(2, (i - 24) / 12.0);
        GuitarString gs(frequency);
        samples[i] = makeSampleFromString(gs);
        if (!soundBuffers[i].loadFromSamples(&samples[i][0], samples[i].size
(), 2, HZ))
            throw std::runtime_error("SoundBuffer: failed to load from samples.");
                sounds[i].setBuffer(soundBuffers[i]);
    }

    while (window.isOpen()) {
        while (window.pollEvent(event)) {
            switch (event.type) {
            case Event::Closed:
                window.close();
                break;
            case Event::TextEntered:
                if (event.text.unicode < 128) {
                    string temp;
                    temp += static_cast<char>(event.text.unicode);
                    int index = keyboard.find(temp);
                    sounds[index].play();
                }
                if (Keyboard::isKeyPressed(Keyboard::Escape)) {
                    window.close();
                    break;
                }
                break;
            default:
                break;
            }
            window.clear();
            window.draw(background);
            window.display();
        }
    }
    return 0;
}
```

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

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

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

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

#include "GuitarString.hpp"

BOOST_AUTO_TEST_CASE(GS) {
  vector<sf::Int16> v;

  v.push_back(0);
  v.push_back(2000);
  v.push_back(4000);
  v.push_back(-10000);

  BOOST_REQUIRE_NO_THROW(GuitarString gs = GuitarString(v));

  GuitarString gs = GuitarString(v);

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

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

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

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

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

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

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

  // a few more times
  gs.tic();
  BOOST_REQUIRE(gs.sample() == -4483);
  gs.tic();
  BOOST_REQUIRE(gs.sample() == 1984);
  gs.tic();
  BOOST_REQUIRE(gs.sample() == 0);
}
```

PS6 Airport Simulation (C++ Concurrency)

In this assignment, I had to simulate landings of multiple airplanes on multiple runways at Logan Airport. Use thread and mutex to run multiple threads of code at the same time, which simulated multiple landing lanes landing at the same time. OO design was mutex, and using switch statements to lock and unlock different runways when they were in use. In this assignment I implemented switch statement to check if lanes are being used by different planes.

I learned how to use thread and mutex libraries to run multiple different threads at the same time. One issue I had was that the first time I ran Airport_Sync the planes crashed after a few seconds, but after that it ran fine, for 15 min. I do not know what caused this, so if this happens when you test it for the first time, try again and it might fix itself. Another problem I had was that my terminal only saves 10000 lines so my output.txt only has 10000 output lines.


adam@adam-XPS-13-9360: ~/Desktop/COMP4/PS6
File Edit View Search Terminal Help

```
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 #6 is taxiing on Runway 9 for 2 milliseconds
Airplane #6 is releasing any needed runway(s) after landing on Runway 9
Airplane #6 is waiting for 66 milliseconds before landing again
Airplane #4 is acquiring any needed runway(s) for landing on Runway 14

Checking airport status for requested Runway 14...
Number of simultaneous landing requests == 1, max == 6
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 #4 is taxiing on Runway 14 for 4 milliseconds
Airplane #4 is releasing any needed runway(s) after landing on Runway 14
Airplane #4 is waiting for 2 milliseconds before landing again
Airplane #5 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 == 6
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 8 milliseconds
Airplane #5 is releasing any needed runway(s) after landing on Runway 15L
Airplane #5 is waiting for 82 milliseconds before landing again
```

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

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

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

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

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

#ifndef AIRPLANE_H
#define AIRPLANE_H

#include "AirportRunways.hpp"
#include "AirportServer.hpp"


class Airplane
{
public:

        int airplaneNum;
        AirportServer* apServ;

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


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


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


private:

        AirportRunways::RunwayNumber requestedRunway; // Picked at random

}; // end class Airplane

#endif

```

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

#include "Airplane.hpp"

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

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

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

                apServ->reserveRunway(airplaneNum, requestedRunway);

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

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

        } // end while

} // end Airplane::land
```

```cpp
/**
 *  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.hpp"


class AirportServer {
    public:

        /**
         * Default constructor for AirportServer class
         */
        AirportServer() {
            // ***** Initialize any Locks and/or Condition Variables here as necessary *****
            lck15L = std::unique_lock < std::mutex > (run15L);
            lck15R = std::unique_lock < std::mutex > (run15R);
            lck4L = std::unique_lock < std::mutex > (run4L);
            lck4R = std::unique_lock < std::mutex > (run4R);
            lck14 = std::unique_lock < std::mutex > (run14);
            lck9 = std::unique_lock < std::mutex > (run9);

        } // 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

    /**
    AirportServer.h Tue Apr 23 19:36:55 2019 2
    * Declarations of mutexes and condition variables
    */
    mutex runwaysMutex; // Used to enforce mutual exclusion for acquiring &
releasing runways
    /**
    * ***** Add declarations of your own Locks and Condition Variables
     here *****
    */
    std::mutex run15L;
    std::mutex run15R;
    std::mutex run4L;
    std::mutex run4R;
    std::mutex run14;
    std::mutex run9;

    std::unique_lock < std::mutex > lck15L;
    std::unique_lock < std::mutex > lck15R;
    std::unique_lock < std::mutex > lck4L;
    std::unique_lock < std::mutex > lck4R;
    std::unique_lock < std::mutex > lck14;
    std::unique_lock < std::mutex > lck9;

    std::condition_variable cv;

}; // end class AirportServer

#endif
```

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

#include "AirportServer.hpp"


/**
 * Called by an Airplane when it wishes to land on a runway
 */
void AirportServer::reserveRunway(int airplaneNum, AirportRunways::RunwayNumber runway) {
    // Acquire runway(s)
    { // Begin critical region

        //unique_lock<mutex> runwaysLock(runwaysMutex);

        {
            unique_lock < mutex > lk(AirportRunways::checkMutex);
            cv.wait(lk, [] {
                return !(AirportRunways::getNumLandingRequests() >= 6);
            });

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

        }

        /**
         * ***** Add your synchronization here! *****
         */
        switch (runway) {
        case AirportRunways::RUNWAY_4L:
            cv.wait(lck4L, [ = ] {
                bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4L] == 0);
                bool av15L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15L] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                if (av4L && av15L && av15R)
                    return true;
                else
                    return false;
            });
            cv.wait(lck15L, [ = ] {
                bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4L] == 0);
                bool av15L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15L] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                if (av4L && av15L && av15R)
                    return true;
                else
                    return false;
            });
            cv.wait(lck15R, [ = ] {
                bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUN
WAY_4L] == 0);
                bool av15L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15L] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                if (av4L && av15L && av15R)
                    return true;
                else
                    return false;
            });
            break;
        case AirportRunways::RUNWAY_4R:
            cv.wait(lck4R, [ = ] {
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15L] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_9] == 0);
                if (av4R && av15L && av15R && av9)
                    return true;
                else
                    return false;
            });
            cv.wait(lck15L, [ = ] {
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15L] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_9] == 0);
                if (av4R && av15L && av15R && av9)
                    return true;
                else
                    return false;
            });
            cv.wait(lck15R, [ = ] {
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15L] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_9] == 0);
                if (av4R && av15L && av15R && av9)
                    return true;
                else
                    return false;
            });
            cv.wait(lck9, [ = ] {
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15L] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_9] == 0);
                if (av4R && av15L && av15R && av9)
                    return true;
                else
                    return false;
            });
            break;
        case AirportRunways::RUNWAY_15R:
            cv.wait(lck4L, [ = ] {
                bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4L] == 0);
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_9] == 0);
                if (av4L && av4R && av15R && av9)
                    return true;
                else
                    return false;
            });
            cv.wait(lck4R, [ = ] {
                bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4L] == 0);
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_9] == 0);
                if (av4L && av4R && av15R && av9)
                    return true;
                else
                    return false;
            });
            cv.wait(lck15R, [ = ] {
                bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4L] == 0);
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_9] == 0);
                if (av4L && av4R && av15R && av9) return true;
                else return false;
            });
            cv.wait(lck9, [ = ] {
                bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4L] == 0);
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_9] == 0);
                if (av4L && av4R && av15R && av9)
                    return true;
                else
```

```cpp
                    return false;
            });
            break;
        case AirportRunways::RUNWAY_15L:
            cv.wait(lck4L, [ = ] {
                bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4L] == 0);
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15L] == 0);
                if (av4L && av4R && av15L)
                    return true;
                else
                    return false;
            });
            cv.wait(lck4R, [ = ] {
                bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4L] == 0);
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15L] == 0);
                if (av4L && av4R && av15L)
                    return true;
                else
                    return false;
            });
            cv.wait(lck15L, [ = ] {
                bool av4L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4L] == 0);
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15L = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15L] == 0);
                if (av4L && av4R && av15L)
                    return true;
                else
                    return false;
            });
            break;
        case AirportRunways::RUNWAY_9:
            cv.wait(lck4R, [ = ] {
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_9] == 0);
                if (av4R && av15R && av9)
                    return true;
                else
                    return false;
            });
            cv.wait(lck15R, [ = ] {
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_9] == 0);
                if (av4R && av15R && av9)
                    return true;
                else
                    return false;
            });
            cv.wait(lck9, [ = ] {
                bool av4R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_4R] == 0);
                bool av15R = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_15R] == 0);
                bool av9 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_9] == 0);
                if (av4R && av15R && av9)
                    return true;
                else
                    return false;
            });
            break;
        case AirportRunways::RUNWAY_14:
            cv.wait(lck4L, [ = ] {
                bool av14 = (AirportRunways::runwayInUse[AirportRunways::RUNWAY_14] == 0);
                if (av14)
                    return true;
                else
                    return false;
            });
            break;
        }
        // Check status of the airport for any rule violations
        AirportRunways::checkAirportStatus(runway);

    } // End critical region

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

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

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

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

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

} // end AirportServer::reserveRunway()

/**
 * Called by an Airplane when it is finished landing
 */
void AirportServer::releaseRunway(int airplaneNum, AirportRunways::RunwayNumber runway) {
    // Release the landing runway and any other needed runways
    { // Begin critical region
        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! *****
         */
        switch (runway) {
        case AirportRunways::RUNWAY_4L:
            run4L.unlock();
            run15L.unlock();
            run15R.unlock();
            break;
        case AirportRunways::RUNWAY_4R:
            run4R.unlock();
            run15L.unlock();
            run15R.unlock();
            run9.unlock();
            break;
        case AirportRunways::RUNWAY_15R:
            run4L.unlock();
            run4R.unlock();
            run15R.unlock();
            run9.unlock();
            break;
        case AirportRunways::RUNWAY_15L:
            run4L.unlock();
            run4R.unlock();
            run15L.unlock();
            break;
        case AirportRunways::RUNWAY_9:
            run4R.unlock();
            run15R.unlock();
            run9.unlock();
            break;
        case AirportRunways::RUNWAY_14:
            run14.unlock();
            break;
        }
        AirportRunways::decNumLandingRequests();
        cv.notify_one();
        // Update the status of the airport to indicate that the landing is comp
        AirportRunways::finishedWithRunway(runway);
        //runwaysLock.unlock();
    } // End critical region

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

```
/**
*  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--;
        }

        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


private:

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

}; // end class AirportRunways

#endif

```

```cpp
#include "AirportRunways.hpp"

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

int AirportRunways::numLandingRequests = 0;

int AirportRunways::maxNumLandingRequests = 0;

mutex AirportRunways::checkMutex;


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

} // end AirportRunways::runwayName()


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

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

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

        requestRunway(requestedRunway);

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

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

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

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

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

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

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

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

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

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

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

#include "AirportServer.hpp"
#include "AirportRunways.hpp"
#include "Airplane.hpp"

using namespace std;


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

} // end run


int main(void)
{
        AirportServer as;

        vector<thread> apths; // Airplane threads

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

                apths.push_back(thread([] (Airplane* ap){
                        ap->land();
                }, ap));
        }

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

        return 0;

} // end main
```

PS7 Kronos Startup

In this assignment, I had to analyze the Kronos Intouch time clock log and use regular expression to parse the log files in order to find the boot times for the device. In order to complete this assignment, I started by looking at the output files for device 5 and tried to make my output look similar to that format. I then used regex101.com to try different regex sequences to find which one worked best at matching the characters that I wanted it to match to.

The only difficulty with this project was that I had to look at the previous log files in order to see how I should format the output files.

```
CC = g++
CFLAGS = -c -g -std=c++11
OBJ = main.o
DEPS =
LIBS = -ansi -pedantic -Wall -Werror -lboost_regex -lboost_date_time
EXE = ps7

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

ps7: main.cpp
	$(CC) $(OBJ) -o

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

```cpp
// Copyright 2019 Adam Baptista

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

using std::cout;
using std::endl;
using std::string;
using std::ifstream;
using std::ofstream;

using boost::regex;
using boost::smatch;
using boost::gregorian::date;
using boost::gregorian::from_simple_string;
using boost::posix_time::ptime;
using boost::posix_time::time_duration;

template <typename T>
int to_int(const T& sm) {
    return atoi(sm.str().c_str());
}

int main(int argc, char* argv[]) {
    smatch match;
    string line, str_boot, str_time, str_date, str_done;;
    ptime time_1, time_2;
    int line_num = 1;
    bool boot = false;


    if (argc != 2) {
        cout << "Invalid # of command lind arguments" << endl;
        return 0;
    }
    ifstream inFile(argv[1], ifstream::in);
    if (!inFile.is_open()) {
        cout << "Unable to open file \"" << argv[1] << "\"" << endl;
        return 0;
    }

    string outFileName(string(argv[1]) + ".rpt");
    ofstream outFile;
    outFile.open(outFileName.c_str());

    str_boot = "(.*log.c.166.*)";
    str_done = "(.*oejs.AbstractConnector:Started SelectChannelConnector.*)"
;
    str_time = "([[:digit:]]{2}):([[:digit:]]{2}):([[:digit:]]{2})";
    str_date = "([[:digit:]]{4})-([[:digit:]]{1,2})-([[:digit:]]{1,2}) ";

    regex re_boot(str_date + str_time + str_boot);
    regex re_done(str_date + str_time + str_done);


    while (getline(inFile, line)) {
        if (regex_match(line, match, re_boot)) {
            if (boot)
                outFile << "**** Incomplete boot **** \n" << endl;
        date _date(from_simple_string(match[0]));
        ptime temp(_date, time_duration(to_int(match[4]), to_int(match[5]),
            to_int(match[6])));
        time_1 = temp;

        outFile << "=== Device boot ===" << endl;
        outFile << line_num << "(" << argv[1] << "): ";
        outFile << match[1] << "-" << match[2] << "-" << match[3] << " ";
        outFile << match[4] << ":" << match[5] << ":" << match[6] << " ";
        outFile << "Boot Start" << endl;
        boot = true;

        } else if (regex_match(line, match, re_done)) {
            if (boot) {
                date _date(from_simple_string(match[0]));
                ptime temp(_date, time_duration(to_int(match[4]),
                    to_int(match[5]), to_int(match[6])));
                time_2 = temp;

                time_duration td = time_2 - time_1;

                outFile << line_num << "(" << argv[1] << "): ";
                outFile << match[1] << "-" << match[2] << "-"
                    << match[3] << " ";
                outFile << match[4] << ":" << match[5] << ":"
                    << match[6] << " ";
                outFile << "Boot Completed" << endl;

                outFile << "\tBoot Time: ";
                outFile << td.total_milliseconds() << "ms \n" << endl;

                boot = false;
            } else {
                outFile << "**** Unexpected boot ****\n" << endl;
            }
        }
        line_num++;
    }
    return 0;
}

```