

Final Portfolio

By: Daniel Alvarez

Table of Contents

Main directory:

Contains this document and subdirectories for each project included in this portfolio. The sub directories and their individual contents are listed below.

- **Ps0** directory contains the files for the sfml hello word project:
 - Main.cpp – the main file of the project
 - Sprite.png – the image used in main.cpp
 - screenshot.png - Screenshot of code running
 - ps0-readme.txt - a readme file
 - Makefile - The makefile for the project
- **Ps1** directory contains the files for the Sierpinski fractal project:
 - Sierpinski.cpp – the main file of the triangle fractal
 - Sierpinski.hpp – header file for Sierpinski.cpp
 - original.cpp – the main file of the square fractal
 - original.hpp – header file for original.cpp
 - original fractal.png – screenshot of the square fractal
 - Sierpinski fractal.png – screenshot of the triangle fractal
 - Readme – a readme file
 - Makefile – the makefile for the project
- **Ps2a** directory contains the files for linear feedback shift register project:
 - LFSR.cpp – contains implementation of the LFSR class
 - LFSR.hpp – the header file for LFSR.cpp
 - Boosttest.cpp – unit tests for LFSR using the boost library
 - Ps2a-readme.txt – a readme file
 - Makefile – the makefile for the project
- **Ps2b** directory contains the files for linear feedback shift register image encryption project:
 - LFSR.cpp – code for the LFSR class
 - LFSR.hpp – header file for the LFSR class
 - photoMagic.cpp – encrypts and decrypts image files using the LFSR class
 - Makefile – the makefile for the project

- **StrBlob** directory contains the files for the smart pointers: StrBlob project:

Note: Initial files were provided by Charles Wilkes, modifications noted below

- StrBlob.cpp – implementation of the StrBlob class StrBlob.hpp – header file for StrBlob.
 - UniqueStrBlob.cpp – implementation of the UniqueStrBlob class.
Modified by Daniel Alvarez
 - UniqueStrBlob.hpp – header file for UniqueStrBlob.cpp. Modified by Daniel Alvarez
 - TestStrBlob.cpp – test driver for StrBlob class
 - TestUniqueStrBlob.cpp – test driver for UniqueStrBlob class
 - Readme – a readme file for the program
 - Makefile – the makefile for the program
- **Airport** directory contains the files for the thread concurrency: Airport simulation project.

Note: Initial files were provided by Charles Wilkes, modifications noted below

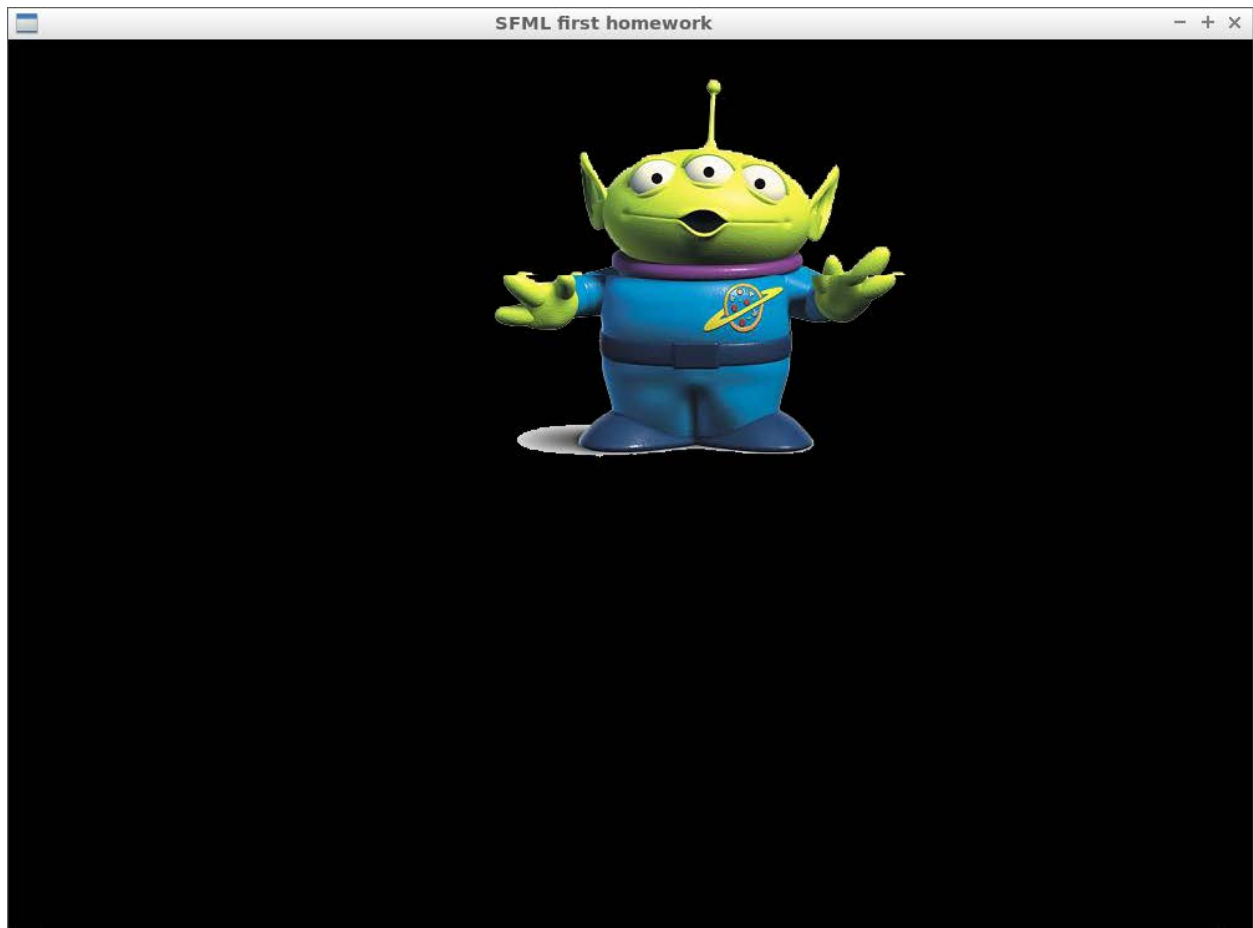
- AirportServer.cpp – code that manages airplane landing requests.
Modified by Daniel Alvarez to add synchronization code.
- AirportServer.hpp – header file for AirportServer.cpp
- Airplane.cpp – definition for the Airplane class
- Airplane.hpp – header file for Airplane.cpp
- AirportRunways.cpp – contains helper methods and constants for the airport simulation
- AirportRunways.hpp – header file for AirportRunways.cpp
- Airport.cpp – airport simulation driver program
- Screenshot.png – a screenshot of the output of the Airport simulation
- Makefile – makefile for the airport simulation

Ps0: SFML Hello World

This project is a simple example of the sfml graphics library. The code in main.cpp first creates a window object, and then loads a texture from a .png file (specifically sprite.png). Then a sprite object is instantiated with the texture. When the program runs the window is displayed and using a while (window.isOpen()) loop the sprite constantly moves horizontally, reversing directions when it reaches the edge of the window. Using two event listeners the program constantly checks to see if either of the vertical arrow keys has been pressed or if the window has been resized. If they have certain sections of code execute depending on the event. The up and down arrows increase or decrease the scale of the sprite and resizing the window rotates the sprite 45 degrees. This program is a small sample of what can be accomplished using the sfml library.

To compile the program: Go to the directory containing the files and enter “g++ -c main.cpp” on the command line. Then enter “g++ main.o -o sfml-app -lsfml-graphics -lsfml-window -lsfml-system”.

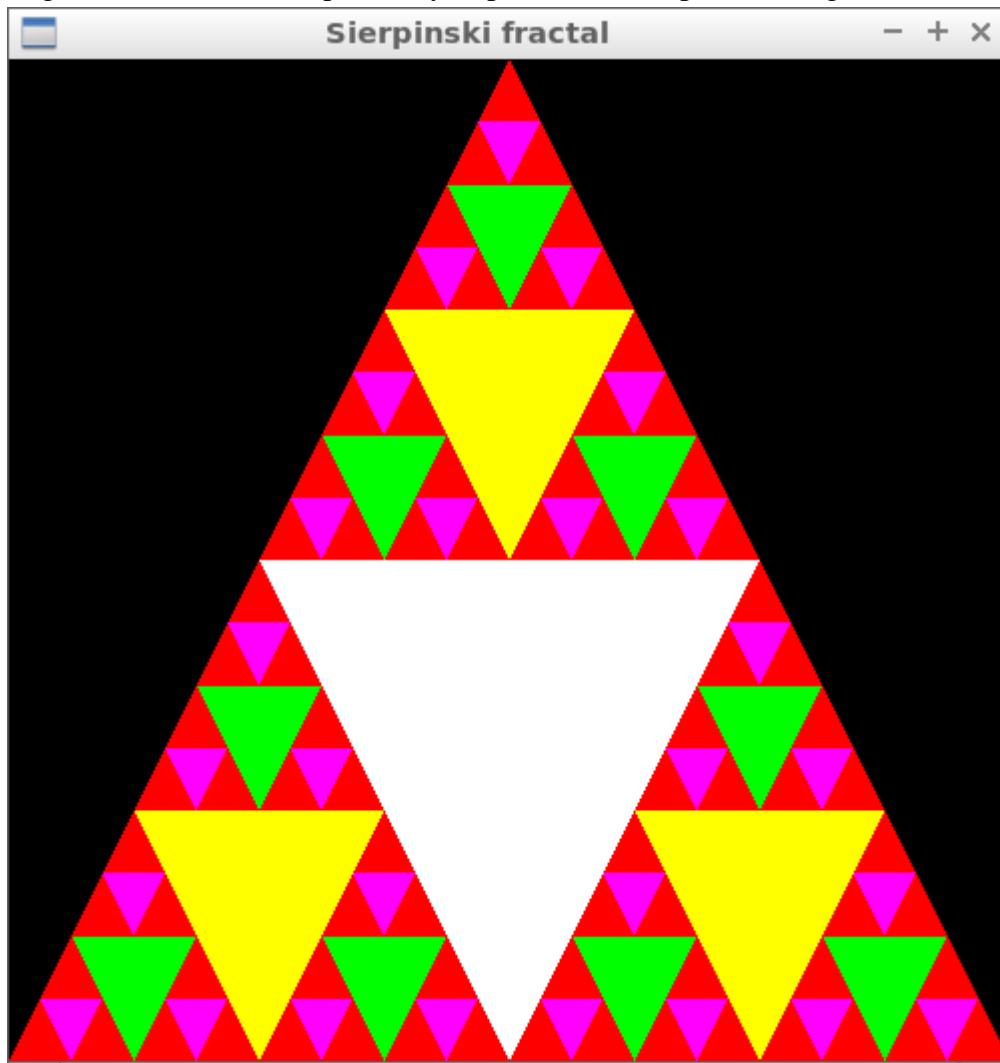
To run the program: Enter “./sfml-app” on the command line.



This is a screenshot of the program as it is running.

Ps1: Fractal generator

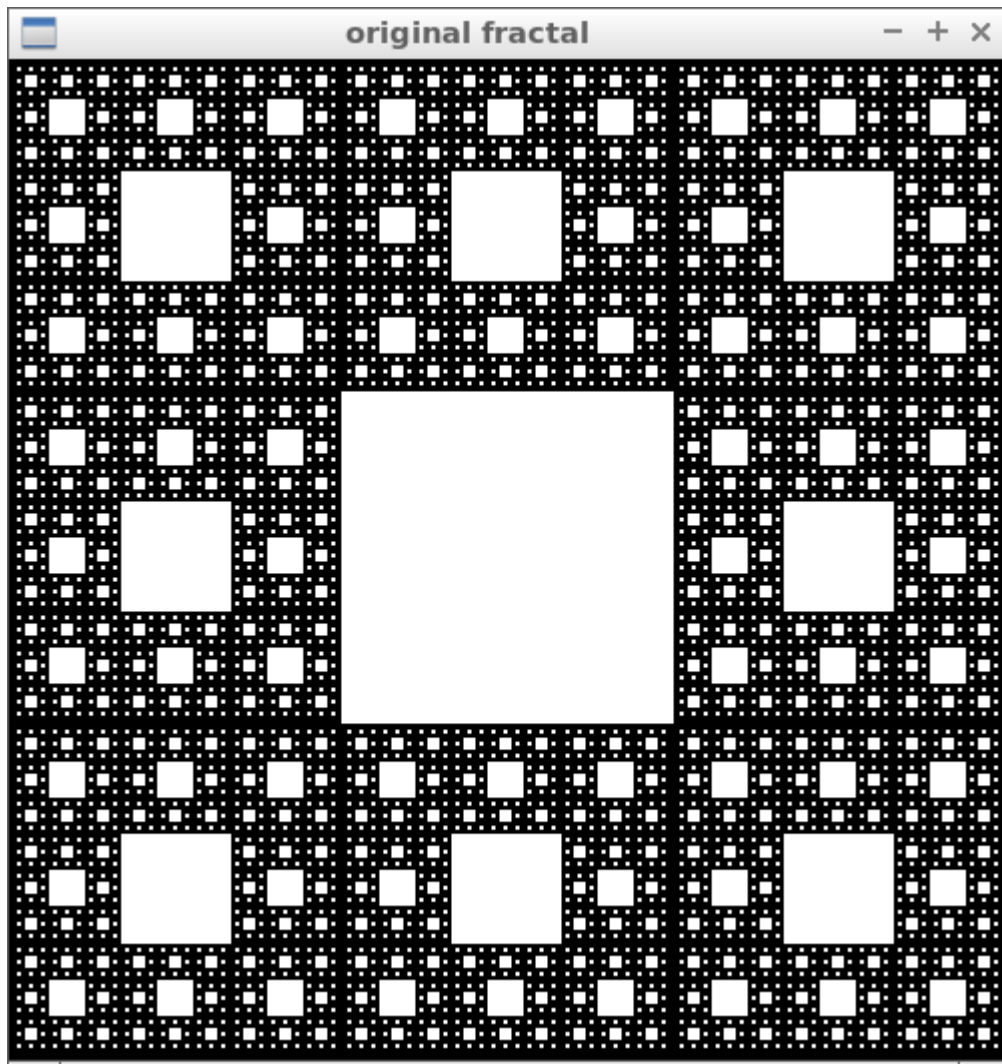
In this project contains two classes that use recursion to repeatedly create and draw shapes in order to display a fractal. The first class, named Sierpinski creates a fractal called a Sierpinski triangle. This fractal is constructed by drawing an equilateral triangle, and then drawing three new equilateral triangles each with side lengths half that of the original in the three corners of the original. Then repeat this process treating each of the three new triangles as the original. This is an example of my implemented Sierpinski triangle fractal



I implemented this using the sfml library to display the triangles and a recursive draw function to create the fractal effect. The draw function takes two parameters, depth and size. The size of the first triangle drawn is determined by the size of the window, later triangle's sizes are

determined by the size of the triangle containing them. Size determines the side length of the triangles and depth is used to keep track of how many layers deep the fractal has left before it stops recursing (leaving it running forever is not an efficient use of computing power). First the draw function draws the “base” triangle, and then it checks to see if depth is zero and if so it stops. Otherwise it decrements depth by one and calls draw three times, once for each smaller triangle inside the base triangle with new size and depth values. This continues until depth reaches zero at which point the program stops drawing new triangles. The different depths are drawn in different colors using a switch statement that is controlled by the depth modulo 5, making the layers cycle through five different colors before repeating.

This project has another class, named original that draws a second type of fractal using a similar process. This fractal starts with a base square, then draws 8 smaller squares around it, and then 8 even smaller squares around each of those and so on as shown below.



The implementation for this fractal works exactly the same as the Sierpinski triangle, but the recursive draw function calls itself eight times instead of three.

To compile this program: Go to the directory containing the files and enter “make all” on the command line.

Then to generate the Sierpinski fractal enter “./Sierpinski [desired recursion depth] [size of window]”. To generate the square fractal instead enter “./original [desired recursion depth] [size of window]”.

Ps2a: Linear Feedback Shift Register

This program produces pseudo random bits by simulating a linear feedback shift register. I used a vector of ints to store my register of bits. In the constructor a for loop grabs each character in the string, converts it to an int, and then adds it to the vector using pushback(). When step() is called it first finds and stores the result of bit[0] XOR bit[tap] (note: bit[0] is the leftmost bit) in a temporary int variable. Then I call the erase(myVector.begin()) to remove the current leftmost bit and use pushback(the_temporary_int_variable) to add the new rightmost bit. I chose to use a vector because they are easy to resize and because using the erase() and pushback() functions means that when I remove the leftmost bit all the other bits have their positions shifted automatically which is much easier for me.

I created two test functions using the boost library in addition to the default ones generated automatically.

Custom test 1:

This test checks that the constructor correctly sets tap to the rightmost bit if the value received is not a valid value for tap.

Custom test 2:

This test Checks that my code functions correctly when the seed is only a single bit. (the code won't provide any useful output, step() will always return a 0, but the code should run successfully and without errors).

To compile this program: Go to the directory containing the files and enter “make all” in the command line.

To run the tests enter “./boosttest” to run the test driver for the program.

Ps2b: Image Encryption using LFSR

This project loads a .png file passed to it from the command line and encrypts the color data of each pixel using the LFSR class from the previous project. It displays both the original image and the encrypted image, and then saves the encrypted image in a second .png file passed in from the command line. The advantage to encrypting the image this way is that if you run the

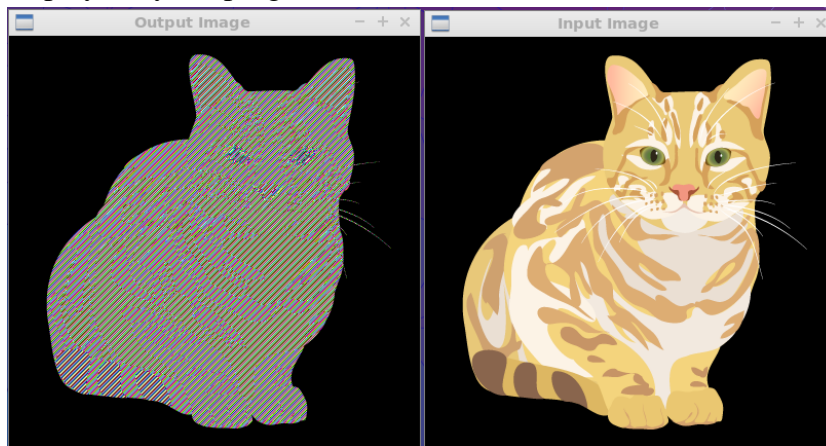
program on the encrypted image using the same LFSR seed and tap number it will give you back the original image. It functions both as an encrypter and a decrypter.

The implementation of this program is as follows:

Parameters: four command line arguments

1. Input file – String
2. Output file – String
3. LFSR seed – String
4. LFSR tap position – int

The program uses sfml to load the image then runs a function I wrote named transform. Transform takes an image and an LFSR object as parameters and returns an image. Inside transform two nested (one to traverse the x axis and one to traverse the y axis) for loops iterate over every pixel in the image and xor the red, green, and blue values with three 8 bit integers generated by the LFSR. The loop saves the new color values to the pixel and then moves to the next pixel in the image. After both loops end and the image is fully transformed the function returns the new, encrypted image. The main function in PhotoMagic.cpp stores the returned image in a new image variable and displays the original and the encrypted versions of the image in two separate sfml windows. Finally it saves the encrypted image. An example of the windows displayed by the program is shown below.



To compile this program: Go to the directory containing the files and enter “make” all in the command line.

To run the encrypter/decrypter: Enter “./PhotoMagic [name of input file] [name of file to output encrypted image to] [LFSR seed] [LFSR tap position]” on the command line.

StrBlob:

The project involved taking a class StrBlob that held a shared_ptr to a vector of strings and creating a new class UniqueStrBlob that had the same functionality, but used a unique_ptr. First I modified the code from StrBlob to use unique pointers and then added a new function and an overloaded operator. The new function I added is called makeSharedBlob and it passes the data from the calling UniqueStrBlob object to a new StrBlob object and leaves the UniqueStrBlob that called it with a nullptr after the function finishes running. Lastly, I overloaded the addition operator for UniqueStrBlob to receive two UniqueStrBlob objects as inputs. It grabs the data from the vector pointed to by the right hand side UniqueStrBlob and append it to the vector pointed to by the left hand side UniqueStrBlob.

To compile this program: Go to the directory containing files and enter “make all” on the command line.

To run this program: Enter “./TestUniqueStrBlob” on the command line.

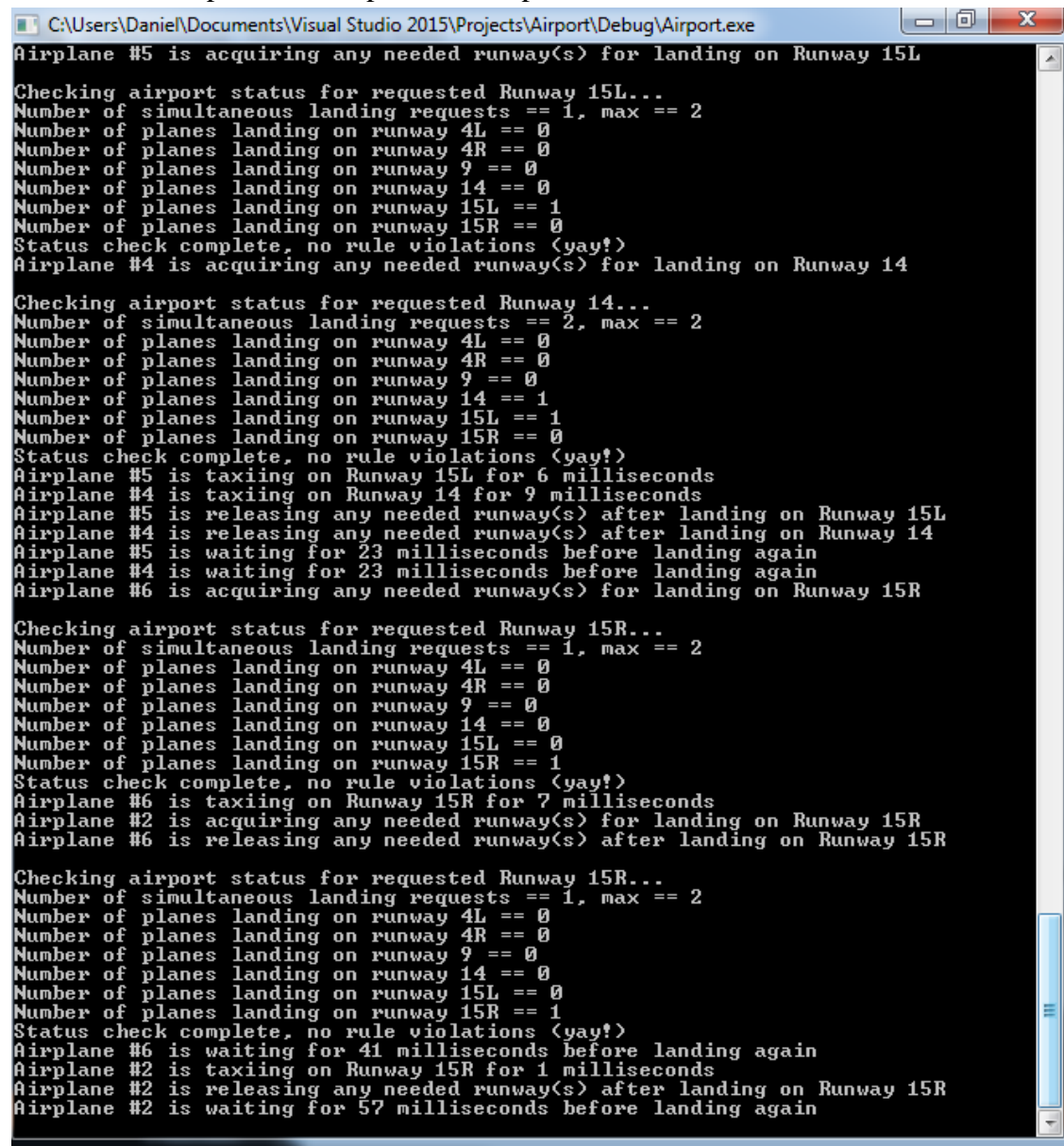
Airport:

This program simulates an air traffic control system that manages plane landings at an airport. The airport has six runways, 4L, 4R 9, 14, 15L, 15R and with the exception of 14 each runway conflicts with at least one other runway, those runways cannot be used simultaneously. This simulation has 7 airplanes, each one represented by a thread. These planes continuously attempt to land at a random runway, upon landing they taxi briefly before magically being back in the sky to make another landing. When a plane is landing the program checks to see if the number of planes on any runway is greater than one or if any runways conflicting with the planes chosen runway are in use. If either of these happened the program displays a plane crashed message and ends, otherwise the plane lands and begins taxiing.

To prevent crashes I used a mutex unique_lock, a condition variable and an array of the locked/unlocked status of the six runways. The mutex lock is used by the program to prevent a plane from landing on a runway after another plane has landed, but before the runway has been locked. Only one thread (airplane) can own the unique lock at a time and a plane must have ownership of the lock to make changes to the status of the runways. To prevent crashing I included two checks in the lambda function of the wait method of the condition variable. First the code checks if the plane’s chosen runway already has a plane on it, if so the plane waits. Next

the code checks if any runways that conflict with the plane's chosen runway are in use by using a switch statement and the array of runway locked/unlocked statuses. If it finds any conflicts the plane waits. If there is no conflict the plane locks its chosen runway and runways that conflict with it, releases ownership of the unique_lock and then lands and starts taxiing. When the plane finishes taxiing and returns to the air it unlocks any runways it was using.

Here is an example of the output of the airport simulation.



```
C:\Users\Daniel\Documents\Visual Studio 2015\Projects\Airport\Debug\Airport.exe
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 == 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 #4 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 == 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 == 1
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 6 milliseconds
Airplane #4 is taxiing on Runway 14 for 9 milliseconds
Airplane #5 is releasing any needed runway(s) after landing on Runway 15L
Airplane #4 is releasing any needed runway(s) after landing on Runway 14
Airplane #5 is waiting for 23 milliseconds before landing again
Airplane #4 is waiting for 23 milliseconds before landing again
Airplane #6 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 #6 is taxiing on Runway 15R for 7 milliseconds
Airplane #2 is acquiring any needed runway(s) for landing on Runway 15R
Airplane #6 is releasing any needed runway(s) after 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 #6 is waiting for 41 milliseconds before landing again
Airplane #2 is taxiing on Runway 15R for 1 milliseconds
Airplane #2 is releasing any needed runway(s) after landing on Runway 15R
Airplane #2 is waiting for 57 milliseconds before landing again
```

To compile this program: Go to the directory containing the files and enter “make all” on the command line.

To run this program: Enter “./Airport” on the command line.