<u>Reflective Paper</u> – Joseph Dzagli

My major observation about c++ during this project is how pointers gave convenience in passing values between functions. The fact that you had to import every file explicitly in any other file where it was used. Our failure to do so troubled us with a great deal of time and energy wasted debugging until we realized this fact. Numspace came in very handy, because having to call it explicitly on every declaration was daunting. Anytime we missed it, we were faced with the tedious challenge of debugging. A major convenience came with the use of vectors, java required that you state the size of an array before it was created, but vectors afforded us the opportunity to expand the size of the array as it was being updated. The use of pointers was quite tricky. The fact that you also had to declare all methods that would be used at the top of the file was an unnecessary feature with c++ which was more convenient with Java, because java did that implicitly in the background.

Generally, the rules of syntax for both c++ and java were the same. Below is a description of how our code works.

The algorithm takes the City and Country from the user request input text file, finds the appropriate airport instance in the airports.csv file and creates an airport object for each relevant instance. Using the AirportID, IATA and or ICAO of a given airport instance it finds valid routes from the routes.csv file from a given source airport to an available destination airport.

The algorithm makes use of a simple heuristic to find the path to a destination airport. It starts at the source airport. It compiles a frontier list of all the airports that it can go to from that state airport. It computes the distance between each of the airports within that frontier list and the destination airport. It chooses to go to the airport which is closest to the destination airport, thus has the least distance away from the destination airport and that airport becomes its new current state. It repeats this process until it arrives at the destination airport. Airports are selected and added to the frontier based on the existence of a valid route between both airports as specified in the routes.csv spreadsheet.

Initially I applied the Pythagoras theorem to calculate the heuristic, however, the earth is a globe and not a plane field so there were errors to its computations, so I decided to apply a formula from the internet to calculate the accurate distance between two geographical locations.

This method is efficient because it is always picking the airport that is closest to the destination airport provided the destination airport is not within direct reach. Since its always picking the closest airport, it would never come to a place where the algorithm would go far away from the destination airport, instead, it goes closer to the destination airport with each step.

I created classes for the airports, routes and Airplanes to enable me access their attributes without reading through the database every time.

I created constructors for each of those classes to take a row in the spreadsheet and convert it into a class, identifying and initialising each of its instances.

I created a method in my main class to receive path search request from the user and store it into a text file. **Request()**

I created another class to read the input text file and feed the parameters into the search algorithm. **ProcessRequest()**

I created a function to implement my heuristic for the search algorithm. **Heurstic()**
I also created functions to create the class of the source airport and the destination airport within my main method. **findAirportClass()** and **findDAirportClass()**

I created a class to find all airports reachable from a current state aiport.

findAllDestinationAirports().

A class to print results and provide final output text file. **printResults().**

The **findRoute()** function returns the route from a given source airport to a destination airport.