|  |
| --- |
| **A picture containing text, clipart  Description automatically generated** |

**Jordan Ayiku Teye**

**40792024**

**CS 313: Intermediate Computer Programming**

**Cohort B**

**Dr. Robert Sowah**

**November 30th, 2022**

The project started off by using classes to represent different aspects of the entire algorithm. An Airport class was made to store the needed information of the Airports such as the ID, name and IATA code. A node class was used to store airports and parent nodes to perform breadth-first-search, this class however proved to be problematic considering the use of pointers. A graph was made considering it would have been the most efficient way to represent and search through connected airports. The main class incorporated all other classes to perform the task at hand.

From the above insight, Felix and I re-started our project by attempting to implement lists to store our Airport Id’s and Airport names but turned out to be a tedious job in c++. We re-started again because we later realized that using vectors and an unordered map would be easier to work with in c++ which is how the graph was represented. We stored the Airport Id’s as the keys in a different unordered map called names to reference Airport using Airport Ids where necessary. Essentially, the code worked by creating a graph of all the Airports by reading the airports.csv file and added edges to each vertex by reading the routes.csv file.

To write the results to a file, a 2 by 2 vector was created to store the start city, start country, destination city and destination. Then a vector containing the values of the start airport ids and destination airport ids was generated. This was necessary considering one country can have multiple airports. Using our vectors start\_airportsiDs and dest\_airportsiDs we searched for direct flights first. If there were no direct flights, then we stored the names of the starting airports and destination airports using start\_airportiDs and dest\_airportIDs into the variables start\_airport and dest\_airport and traversed the bfs graph for those starting airports and destination airports that were not in the vectors start\_airportsiDs and dest\_airportsiDs.

Unfortunately we couldn’t get the program to find airports that weren’t direct because our node class used pointers and caused segmentation errors which we couldn’t fix. That was the only part of the code that seems to be the issue in outputting the appropriate values.