# READMEs:

README: for NearestAirport.tar.gz

To Unpackage:

-run: "tar -zxvf NearestAirport.tar.gz"

-produces folder: GreatCircle

To Make/compile:

-run command: Make

To Execute:

-run: ./NearestAirport

During Execution:

-User input: done in a [latitude] [longitude] [altitude]

-ex: 32 -97 10

-output: 5 nearest airports with distance in miles

-to exit program: -1 -1 -1

To try out tests:

-Test1: time ./NearestAirport < Test1.dat

-Test2: time ./NearestAirport < Test2.dat

-Test3: time ./NearestAirport < Test3.dat

-Results:

- gives an output of:

real: time in \_min and \_.\_ \_ \_sec

user: time in \_min and \_.\_ \_ \_sec

sys: time in \_min and \_.\_ \_ \_sec

-where:

-real: is the wall clock time, from start to finish from time you hit

the enter key to when the wget command is completed

-user: time spent in user mode

-sys: time spent in kernel mode

------------------------------------------------------------------------------------------------------------------------------------------

README: for NearestAirportWithClock.tar.gz

To Unpackage:

-run: "tar -zxvf NearestAirportWithClock.tar.gz"

-produces folder: GreatCircleWithClock

To Make/compile:

-run command: Make

To Execute:

-run: ./NearestAirport

During Execution:

-User input: done in a [latitude] [longitude] [altitude]

-ex: 32 -97 10

-output: 5 nearest airports with distance in miles

-to exit program: -1 -1 -1

To try out tests:

-Test1: ./NearestAirport < Test1.dat

-Test2: ./NearestAirport < Test2.dat

-Test3: ./NearestAirport < Test3.dat

-Results:

-displays time spent from start of main to end of main.

-Test4: ./NearestAirport < Test4.dat

-Results:

-displays time spent from start of main to end of main. but from multiple points

------------------------------------------------------------------------------------------------------------------------------------------

# Problem Statement:

* Analyze, design, and implement a closest airport program. The User should enter coordinates in “Latitude Longitude Altitude” format, then the program will output the five closest airports from a “airports.dat” file with the name and distance using the great circle algorithm. The program will continue to run until a coordinate of -1 -1 -1 is entered and exit. The program will run via command line with standard input, or Unix shell redirection. The program should output the correct answer, have low variability in the time required to calculate the answer, and use the most efficient implementation with respect to CPU time.
* The programmer will be restricted to Not use C++ STL routines or C Library sorting and searching functions. The programmer should not calculate the distance from all possible airports. If Dynamic memory allocation is to be used, the programmer may only use it before the first user input. The Code should be compiled with -O0.

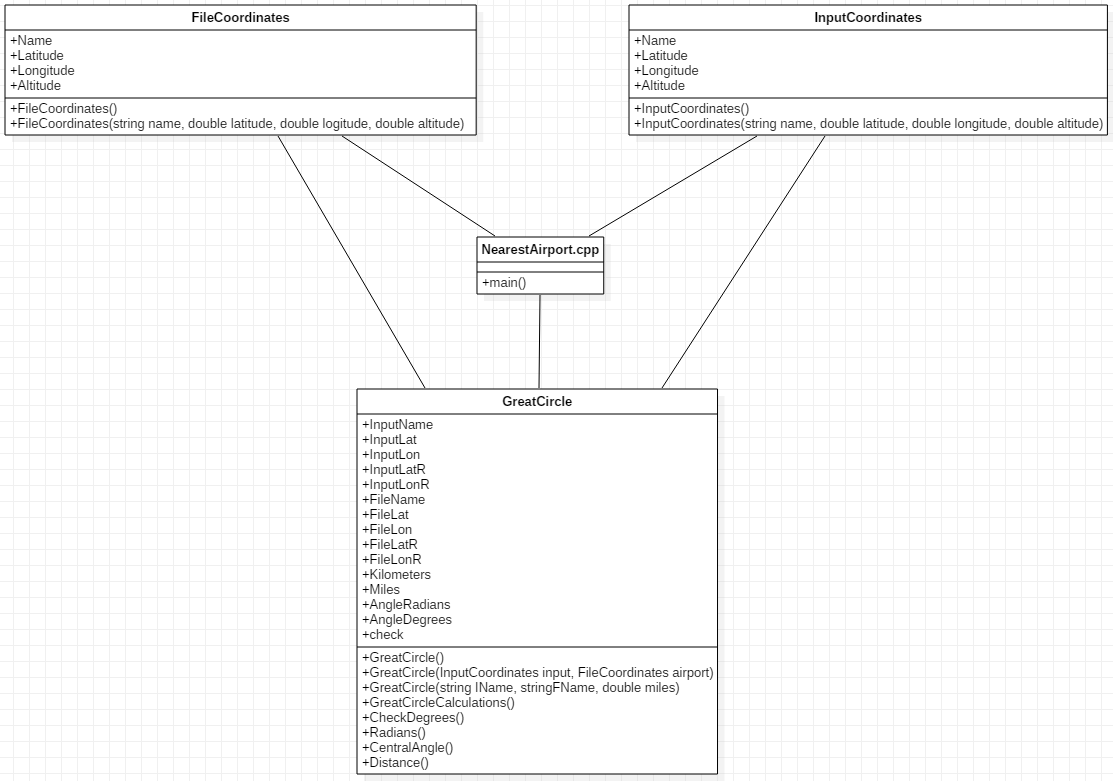
# Schedule:

* Reading assignment: 10 minutes (did this before getting to schedule)
* Analysis of problem: 10 minutes (did this before getting to schedule)
* Problem statement: 10 minutes (did this before getting to schedule)
* Schedule:
  + Assumed: 10 minutes
  + Actual: 9 minutes
* Analysis of requirements:
  + Assumed: 30-45 minutes
  + Actual: 36 minutes
* Researching Primary solutions:
  + Assumed: 30 minutes – 1 hour
  + Actual: 45 minutes
* Design:
  + Assumed: 45 minutes – 1 hour
  + Actual: 1 hour and 30 minutes
* Researching Secondary solutions:
  + Assumed: 30 minutes – 1 hour
  + Actual: 45 minutes
* Alternative Designs:
  + Assumed: 1 hour – 1 hour and 30 minutes
  + Actual: 30 minutes
* Constraints Write up:
  + Assumed: 30 minutes
  + Actual: 25 minutes
* Code Implementation:
  + Assumed: 3 hour – 4 hours
  + Actual: 7 hours
* Testing:
  + Assumed: 30 minutes – 1 hour
  + Actual: 45 minutes
* Analysis of runtime behavior:
  + Assumed: 45 minutes – 1 hour
  + Actual: 1 hour
* Totals (the top three items of this list are added to both assumed and actual):
  + Assumed: 8 hours and 40 minutes – 12 hours and 25 minutes
  + Actual: 13 hours and 55 minutes

# Requirements Analysis:

* Functional:
  + The system shall get a coordinate from the user in a “latitude longitude altitude” format.
  + The system shall get airport data from a file called airports.dat.
  + The system shall print the five closest airports to the coordinate given.
  + The systems shall exit on input of a “-1 -1 -1”.
  + The system shall run on the Unix command line, and input should support Unix shell redirection.
  + The system shall use the great circle algorithm as a reference.
  + The system shall not use the C++ STL routines and C library sorting and searching functions.
  + The system should be compiled with -O0 so that timing will be easier to measure.
  + The system shall not use a calculation of the distance from all possible airports.
  + They system shall not use Dynamic memory allocation after the user adds input.
* Non-Functional:
  + The system shall output the correct answer.
  + The system shall have low variability in the time required to calculate the answer.
  + The system shall use the most efficient implementation with respect to CPU time.

# Design:



The system should have three objects (FileCoordinates, InputCoordinates, and GreatCircle) and one main/driver-program (NearestAirport.cpp).

FileCoordinates & InputCoordinates:

* Description: Both are pretty much the same, but FileCoordinates is used to hold the data from the airport.dat file, and InputCoordinates holds data from the User.
* Attributes: both have a”std::string Name” to identify, and “doubles” for “Latitude, Longitude, and Altitude”.
* Methods:
  + Constructor: both have two constructors, one empty and another for setting the attributes.

GreatCircle:

* Description: holds relevant information from the two Coordinates to be compared, along with attributes to hold the data calculated from the methods.
* Attributes:
  + Input{Name, Lat, Lon, LatR, LonR}, and File{Name, Lat, Lon, LatR, LonR} both are the data holders for the data from InputCoordinates and FileCoordinates and for the conversion of the coordinates in radians.
  + Kilometers and Miles, holds the distance calculated in Kilometers and Miles respectively.
  + AngleRadians AngleDegrees, holds the central angle calculated in radians and Degrees respectively
  + Check, validates that the Input and File coordinates given are valid for the calculations
* Methods
  + Constructor: this class has three constructors an empty one, one to take in InputCoordinates and FileCoordinates for their attributes, and the last constructor just holds the names of the coordinates compared and distance in miles for the final output.
  + Methods:
    - GreatCircleCalculations(): a function that makes a single call for CheckDegrees(), Radians(), CentralAngle(), and Distance()
    - CheckDegrees(): checks “this” objects coordinates to make sure that the points are valid inputs.
    - Radians(): converts the initial coordinates into radians for File and Input LatR and LonR
    - CentralAngle(): calculates the central angle and sets the AngleRadians and AngleDegrees from the radian conversion in the method above.
    - Distance(): calculates the Distance between the two points from the central angle calculation in the method above and sets the Kilometres distance and Miles Distance.

NearestAirport.cpp:

* Description: contains the main() program that will read the file, set the objects listed above, run the calculations, check for the 5 closest airports , and output the 5 found.
* Functionality In order:
  + File Reading: The airport.dat file is read twice, the first time to find the number airports in the file (number of lines), the second to parse the file then set an array of FileCoordinate Airports[].
  + User Input: stuck in a forever while loop that will get the user input and check to see if the user enters -1 -1 -1 and break. Other wise checks the User input for Latitude Longitude and Altitude, then sets them to InputCoordinates Input.
  + Calculations: Takes the results of the two previous functionalities, runs a calculation between the two coordinates for each FileCoordinate and stores it in an array of GreatCircle GreatCircles[].
  + Finding Closest Airports: initializes an array of GreatCircle Closest[5] with the first 5 GreatCircles created in the previous functionality. Then for every set of GreatCircles we check and update the Closest array with a bunch of if else statements to see if there are smaller values in this array. Finally, we print the 5 closest airports from the Closest array.

Run Time Behavior:

* I expect the runtime to run in time O(n) on each loop needed due to no nested loops running (besides the forever “while(1)” which does the work for a single set of user input). So the program will run pretty fast with very little variability and definitely not take anywhere near double the time between two of the same inputs.

# Alternatives:

Alternative Design 1: Accounting for Altitude

* I though about applying the Altitude from the input and from the airport.dat but most readings found online during the research done doesn’t care or shows that altitude doesn’t really contribute to a significant amount(realistically), so this wasn’t included in the calculation’s portions of the design.

Alternative Design 2: Using a single class for Coordinates instead of InputCoordinates and FileCoordinates.

* This would work because the both are the same besides naming. But I needed another Alternative Design to put here, so here we are.

# Constraints:

Given from Requirements (ones significant in coding for me):

* Not using Vectors, or searching/sorting algorithms, using c/c++, No Dynamic memory allocations after user input. So, the program avoids all that.

Given from the airport.dat file:

* Buffers need to be used in parsing the file for the name and coordinates. This is due to the file containing names and cities with commas in them (most values are comma delaminated, so this caused issues initially running the program)

User Input:

* Taking in Altitude as input, though it doesn’t really contribute to most instances of the Great Circle Algorithm. So, we take them in but don’t really do anything with it.

Calculations:

* Some data may be corrupt, or the user input may be invalid coordinates. So a check of the coordinates is done.
* Most instances of the great circle examples found online show the use of the metric system. So, conversions are necessary (degrees to radians, and kilometers to miles).

# Source Code:

Transferred over from visual studio code to aid in visuals.

It is important to note that there are two tar files submitted with two separate README’s to see the difference in time results.

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////NearestAirport.cpp//////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <math.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

#include "GreatCircle.hpp"

int main()

{

    int count = 0;

    // READ IN FILE airport.dat //

    std::ifstream AirportFile;

    AirportFile.open("airports.dat");

    if(!AirportFile.is\_open())

    {

        std::cout << "Could not open the file, Exiting" << std::endl;

        return 1;

    }

    // get the number of inputs we need to allocate for //

    std::string lineCount;

    while(std::getline(AirportFile,lineCount))

    {

        count++;

    }

    AirportFile.close();

    // doing the read in again, because errors if i dont //

    // READ IN FILE airport.dat //

    AirportFile.open("airports.dat");

    if(!AirportFile.is\_open())

    {

        std::cout << "Could not open the file, Exiting" << std::endl;

        return 1;

    }

    // TO STORE OUR AIRPORTS //

    FileCoordinates Airports[count];

    std::string line;

    int i = 0;

    while(std::getline(AirportFile,line))

    {

        std::stringstream ss(line);

        // everything on a line in order //

        std::string AirportId,

                    Name, City, Country, IATA, ICAO,

                    latS, lonS, altS,

                    Timezone, DST, Tz, Type, Source;

        std::string buffer;

        // getting everything from the file line //

        std::getline(ss,AirportId,'\"');

        std::getline(ss,Name,'\"'); // need this

        std::getline(ss,buffer,','); //helps with commas in the name

        std::getline(ss,buffer,'\"'); //helps with commas in the name

        std::getline(ss,City,'\"');

        std::getline(ss,buffer,','); //helps with commas in the name

        std::getline(ss,Country,',');

        std::getline(ss,IATA,',');

        std::getline(ss,ICAO,',');

        std::getline(ss,latS,','); // need this

        std::getline(ss,lonS,','); // need this

        std::getline(ss,altS,','); // need this

        std::getline(ss,Timezone,',');

        std::getline(ss,DST,',');

        std::getline(ss,Tz,',');

        std::getline(ss,Type,',');

        std::getline(ss,Source,',');

        double lat = std::stod(latS);

        double lon = std::stod(lonS);

        double alt = std::stod(altS);

        FileCoordinates Airport(Name, lat, lon, alt);

        Airports[i].Name = Airport.Name;

        Airports[i].Latitude = Airport.Latitude;

        Airports[i].Longitude = Airport.Longitude;

        Airports[i].Altitude = Airport.Altitude;

        i++;

    }

    AirportFile.close();

    // Getting User Input //

    while(1)

    {

        // USER INPUT VARS //

     double InputLongitude;

     double InputLatitude;

     double InputAltitude;

     std::string InputName = "User Input";

     // PROMPT USER //

     std::cout << "Enter coordinates Latitude Longitude Altitude" << std::endl;

     std::cin >> InputLatitude >> InputLongitude >> InputAltitude;

     // Leave Program if all input == -1 //

     if(InputLatitude == -1 && InputLongitude == -1 && InputAltitude == -1)

     {

         break;

     }

     // construct input //

     InputCoordinates Input(InputName, InputLatitude, InputLongitude, InputAltitude);

     GreatCircle GreatCircles[count];

     for(int i = 0; i < count; i++)

     {

         // taking two coordinates and running calculations //

         GreatCircle greatcircle(Input, Airports[i]);

         greatcircle.GreatCircleCalculations();

         GreatCircles[i].InputName = greatcircle.InputName;

         GreatCircles[i].FileName = greatcircle.FileName;

         GreatCircles[i].Miles = greatcircle.Miles;

     }

     // this object will be used to check for the closest //

     // setting object to initialize //

     GreatCircle Closest[5]

     {

         {GreatCircles[0].InputName , GreatCircles[0].FileName, GreatCircles[0].Miles},

         {GreatCircles[1].InputName , GreatCircles[1].FileName, GreatCircles[1].Miles},

         {GreatCircles[2].InputName , GreatCircles[2].FileName, GreatCircles[2].Miles},

         {GreatCircles[3].InputName , GreatCircles[3].FileName, GreatCircles[3].Miles},

         {GreatCircles[4].InputName , GreatCircles[4].FileName, GreatCircles[4].Miles},

     };

     for(int j = 0; j < count; j++)

     {

         if(GreatCircles[j].Miles < Closest[0].Miles)

         {

             //update last element

             Closest[4].InputName = Closest[3].InputName;

             Closest[4].FileName = Closest[3].FileName;

             Closest[4].Miles = Closest[3].Miles;

             //update 4th element

             Closest[3].InputName = Closest[2].InputName;

             Closest[3].FileName = Closest[2].FileName;

             Closest[3].Miles = Closest[2].Miles;

             //update 3rd element

             Closest[2].InputName = Closest[1].InputName;

             Closest[2].FileName = Closest[1].FileName;

             Closest[2].Miles = Closest[1].Miles;

             //update 2nd element

             Closest[1].InputName = Closest[0].InputName;

             Closest[1].FileName = Closest[0].FileName;

             Closest[1].Miles = Closest[0].Miles;

             // update the first element //

             Closest[0].InputName = GreatCircles[j].InputName;

             Closest[0].FileName = GreatCircles[j].FileName;

             Closest[0].Miles = GreatCircles[j].Miles;

         }

         else if(GreatCircles[j].Miles < Closest[1].Miles || GreatCircles[j].Miles == Closest[0].Miles)

         {

             //update last element

             Closest[4].InputName = Closest[3].InputName;

             Closest[4].FileName = Closest[3].FileName;

             Closest[4].Miles = Closest[3].Miles;

             //update 4th element

             Closest[3].InputName = Closest[2].InputName;

             Closest[3].FileName = Closest[2].FileName;

             Closest[3].Miles = Closest[2].Miles;

             //update 3rd element

             Closest[2].InputName = Closest[1].InputName;

             Closest[2].FileName = Closest[1].FileName;

             Closest[2].Miles = Closest[1].Miles;

             //update 2nd element

             Closest[1].InputName = GreatCircles[j].InputName;

             Closest[1].FileName = GreatCircles[j].FileName;

             Closest[1].Miles = GreatCircles[j].Miles;

         }

         else if(GreatCircles[j].Miles < Closest[2].Miles || GreatCircles[j].Miles == Closest[1].Miles)

         {

             //update last element

             Closest[4].InputName = Closest[3].InputName;

             Closest[4].FileName = Closest[3].FileName;

             Closest[4].Miles = Closest[3].Miles;

             //update 4th element

             Closest[3].InputName = Closest[2].InputName;

             Closest[3].FileName = Closest[2].FileName;

             Closest[3].Miles = Closest[2].Miles;

             //update 3rd element

             Closest[2].InputName = GreatCircles[j].InputName;

             Closest[2].FileName = GreatCircles[j].FileName;

             Closest[2].Miles = GreatCircles[j].Miles;

         }

         else if(GreatCircles[j].Miles < Closest[3].Miles || GreatCircles[j].Miles == Closest[2].Miles)

         {

             //update last element

             Closest[4].InputName = Closest[3].InputName;

             Closest[4].FileName = Closest[3].FileName;

             Closest[4].Miles = Closest[3].Miles;

             //update 4th element

             Closest[3].InputName = GreatCircles[j].InputName;

             Closest[3].FileName = GreatCircles[j].FileName;

             Closest[3].Miles = GreatCircles[j].Miles;

         }

         else if(GreatCircles[j].Miles < Closest[4].Miles || GreatCircles[j].Miles == Closest[3].Miles)

         {

             //update last element

             Closest[4].InputName = GreatCircles[j].InputName;

             Closest[4].FileName = GreatCircles[j].FileName;

             Closest[4].Miles = GreatCircles[j].Miles;

         }

     }

        // Display result //

     std::cout << "The closest 5 airports are:" << std::endl;

     std::cout << "1st: " << "from " << Closest[0].InputName << " to " << Closest[0].FileName << " with a distance: " << Closest[0].Miles << " Miles" << std::endl;

        std::cout << "2nd: " << "from " << Closest[1].InputName << " to " << Closest[1].FileName << " with a distance: " << Closest[1].Miles << " Miles" << std::endl;

        std::cout << "3rd: " << "from " << Closest[2].InputName << " to " << Closest[2].FileName << " with a distance: " << Closest[2].Miles << " Miles" << std::endl;

        std::cout << "4th: " << "from " << Closest[3].InputName << " to " << Closest[3].FileName << " with a distance: " << Closest[3].Miles << " Miles" << std::endl;

        std::cout << "5th: " << "from " << Closest[4].InputName << " to " << Closest[4].FileName << " with a distance: " << Closest[4].Miles << " Miles" << std::endl << std::endl;

    }

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////Coordinates.hpp/////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <string>

#include <iostream>

#include <math.h>

// CLASS FOR USER INPUT //

class InputCoordinates

{

public:

    //constructor//

    InputCoordinates()

    {

    }

    InputCoordinates(std::string name, double latitude, double longitude, double altitude)

    {

        Name = name;

        Latitude = latitude;

        Longitude = longitude;

        Altitude = altitude;

    }

    //atributes//

    std::string Name;

    double Latitude;

    double Longitude;

    double Altitude;

};

// CLASS FOR FILE INPUT //

class FileCoordinates

{

public:

    //constructor//

    FileCoordinates()

    {

    }

    FileCoordinates(std::string name, double latitude, double longitude, double altitude)

    {

        Name = name;

        Latitude = latitude;

        Longitude = longitude;

        Altitude = altitude;

    }

    //atributes//

    std::string Name;

    double Latitude;

    double Longitude;

    double Altitude;

};

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////GreatCircle.cpp//////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <string>

#include <iostream>

#include <math.h>

#include "Coordinates.hpp"

class GreatCircle

{

public:

    // constructor //

    GreatCircle()

    {

    }

    GreatCircle(InputCoordinates input, FileCoordinates airport)

    {

        InputName = input.Name;

        InputLat = input.Latitude;

        InputLon = input.Longitude;

        FileName = airport.Name;

        FileLat = airport.Latitude;

        FileLon = airport.Longitude;

    }

    GreatCircle(std::string IName, std::string FName, double miles)

                : InputName(IName), FileName(FName), Miles(miles)

                {

                }

    // attributes from input //

    std::string InputName;

    double InputLat;

    double InputLon;

    double InputLatR;

    double InputLonR;

    // attributes from file //

    std::string FileName;

    double FileLat;

    double FileLon;

    double FileLatR;

    double FileLonR;

    double Kilometres;

    double Miles;

    double AngleRadians;

    double AngleDegrees;

    bool check;

    // Methods //

    void GreatCircleCalculations()

    {

        CheckDegrees();

        if(check)

        {

            Radians();

    CentralAngle();

    Distance();

        }

    }

    void CheckDegrees()

    {

     check = true;

     if(InputLat < -90.0 || InputLat > 90.0)

     {

     check = false;

     }

     if(InputLon < -180.0 || InputLon > 180.0)

     {

     check = false;

     }

     if(FileLat < -90.0 || FileLat > 90.0)

     {

     check = false;

     }

     if(FileLon < -180.0 || FileLon > 180.0)

     {

     check = false;

     }

    }

    void Radians()

    {

        double DegreesInRadian = 57.29577951;

     InputLatR = InputLat / DegreesInRadian;

     InputLonR = InputLon / DegreesInRadian;

     FileLatR = FileLat / DegreesInRadian;

     FileLonR = FileLon / DegreesInRadian;

    }

    void CentralAngle()

    {

     double LongitudesDifference;

     double DegreesInRadian = 57.29577951;

     if(InputLonR > FileLonR)

     {

     LongitudesDifference = InputLonR - FileLonR;

     }

     else

     {

     LongitudesDifference = FileLonR - InputLonR;

     }

     AngleRadians = acos(sin(InputLatR) \* sin(FileLatR) + cos(InputLatR) \* cos(FileLatR) \* cos(LongitudesDifference));

     AngleDegrees = AngleRadians \* DegreesInRadian;

    }

    void Distance()

    {

        double EarthRadiusKm = 6371;

        double KmPerMile = 1.60934;

     Kilometres = EarthRadiusKm \* AngleRadians;

     Miles = Kilometres / KmPerMile;

    }

};

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////makefile//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

CXX=g++

CXXFLAGS = -Wall -O0 -g -std=c++11

all:NearestAirport

NearestAirport.o:NearestAirport.cpp Coordinates.hpp GreatCircle.hpp

NearestAirport: NearestAirport.o

    ${CXX} -o NearestAirport NearestAirport.o

clean:

    -rm -f NearestAirport NearestAirport.o

# Analysis:

Special Notes:

* It is important to note that there are two tar files submitted with two separate README’s to see the difference in time results.
* It should also be relevant to give my system specifications on what I’m running on to show how I’m getting the results displayed below if it seems either too fast or too slow.
  + OS:
    - Default: Windows 10 (most recent release)
    - VM(where code was made): Linux Mint 19.1 Tessa 64-bit Cinnamon
      * 3 dedicated cores
      * 8192 MB memory
      * 20GB dedicated storage
      * Ran on Oracle VM VirtualBox
  + HardWare:
    - Drives:
      * WD 500GB NVMe PCIe SSD (files here)
      * Samsung 512GB NVMe PCIe SSD 970 PRO
    - RAM
      * (16GB DDR4 2666MHz) x2 ---- 32GB
    - CPU:
      * I7-8750H 2.2-4.1GHz
    - GPU:
      * Nvidia RTX 2080 max-Q

Tests Notes:

Three types of tests were conducted:

* The first is found in the “GreatCircle” folder or the “NearestAirport.tar.gz” using the “time” from the command line.
* When executing:

“Time ./NearestAirport < Test1.dat” or Test2.dat or Test3.dat

This would display:

* + real: time in \_min and \_.\_ \_ \_sec
  + user: time in \_min and \_.\_ \_ \_sec
  + sys: time in \_min and \_.\_ \_ \_sec

where:

* + real: is the wall clock time, from start to finish from time you hit the enter key to when the wget command is completed
  + user: time spent in user mode
  + sys: time spent in kernel mode

The purpose of this was to show the specs of how well the program works on different systems, however these results can show some skewing based on what the operating system is doing before or during execution.

* What is tested:
  + Test1.dat: testing single coordinates 32 -97 10 (should be somewhere in Texas)
    - Output should be airports in Texas
    - Result: Results were airports in that area.
  + Test2.dat: testing single coordinates 51 0 10 (should be somewhere in London)
    - Output should be airports in London or coast of France
    - Result: Results show what it should be.
  + Test3.dat: testing single coordinates 41 2 10 (should be somewhere off the coast of spain)
    - Output should be airports in Spain or off the Illes Balears
    - Result: follows the results from what it should be.
* Results:

|  |  |  |  |
| --- | --- | --- | --- |
| Tests: | real | user | sys |
| 1 | 0m 0.050s | 0m 0.046s | 0m 0.004s |
| 2 | 0m 0.039s | 0m 0.039s | 0m 0.000s |
| 3 | 0m 0.043s | 0m 0.041s | 0m 0.000s |

The results are pretty consistent across the board, but the precision isn’t too high and results may vary based on what the OS is doing. The results also abide by the assumptions made in the Design.

* The second test is available in the second tar file given “GreatCircleWithClock” or “NearestAirportWithClock.tar.gz”

And just executing:

“./NearestAirport < Test1.dat”

Displaying the Results of the program and:

* + Time taken: \_. \_ \_ \_ \_ \_ \_ seconds

The purpose of this test is to see specifically how long the program takes to run from beginning of the main() program to the end of main(). This was done by using clock() from time.h .

* + Used this method because it seemed easier than clock\_gettime(), and it was the first result when I looked it up. Given: <https://www.geeksforgeeks.org/how-to-measure-time-taken-by-a-program-in-c/>
* What was tested:
  + Same as first test.
* Results:

|  |  |
| --- | --- |
| Test | Time from beginning to end |
| 1 | 0.037328 seconds |
| 2 | 0.038856 seconds |
| 3 | 0.036937 seconds |

The results are even more consistent and very similar with a higher precision and very much reflect assumptions made in the design.

* The third test is available in the second tar file given “GreatCircleWithClock” or “NearestAirportWithClock.tar.gz”

And just executing:

“./NearestAirport < Test4.dat”

Displaying the Results of the program and:

* + Time taken: \_. \_ \_ \_ \_ \_ \_ seconds

The purpose of this test is to see specifically how long the program takes to run from beginning of the main() program to the end of main(). This was done by using clock() from time.h .

* + Used this method because it seemed easier than clock\_gettime(), and it was the first result when I looked it up. Given: <https://www.geeksforgeeks.org/how-to-measure-time-taken-by-a-program-in-c/>
* What was tested:
  + Multiple coordinates (the three tested in the first and second test, but in one .dat file). But with multiple attempts of the same result and noting the times (5 tests).
* Results

|  |  |
| --- | --- |
| Test | Time from beginning to end |
| 1 | 0.05065 |
| 2 | 0.050561 |
| 3 | 0.0502 |
| 4 | 0.052842 |
| 5 | 0.048995 |

The results here are very similar to the ones in the second test in terms of variability, but do show a lengthier time than running a single set of coordinates on the list of airports. The results still very much reflect assumptions made in the design.