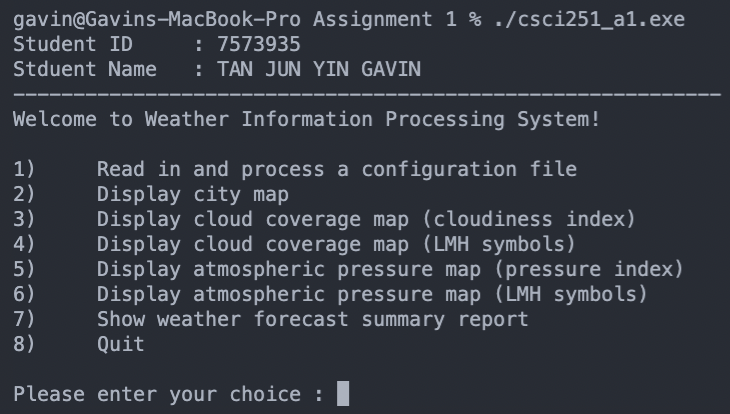
# **Requirements**

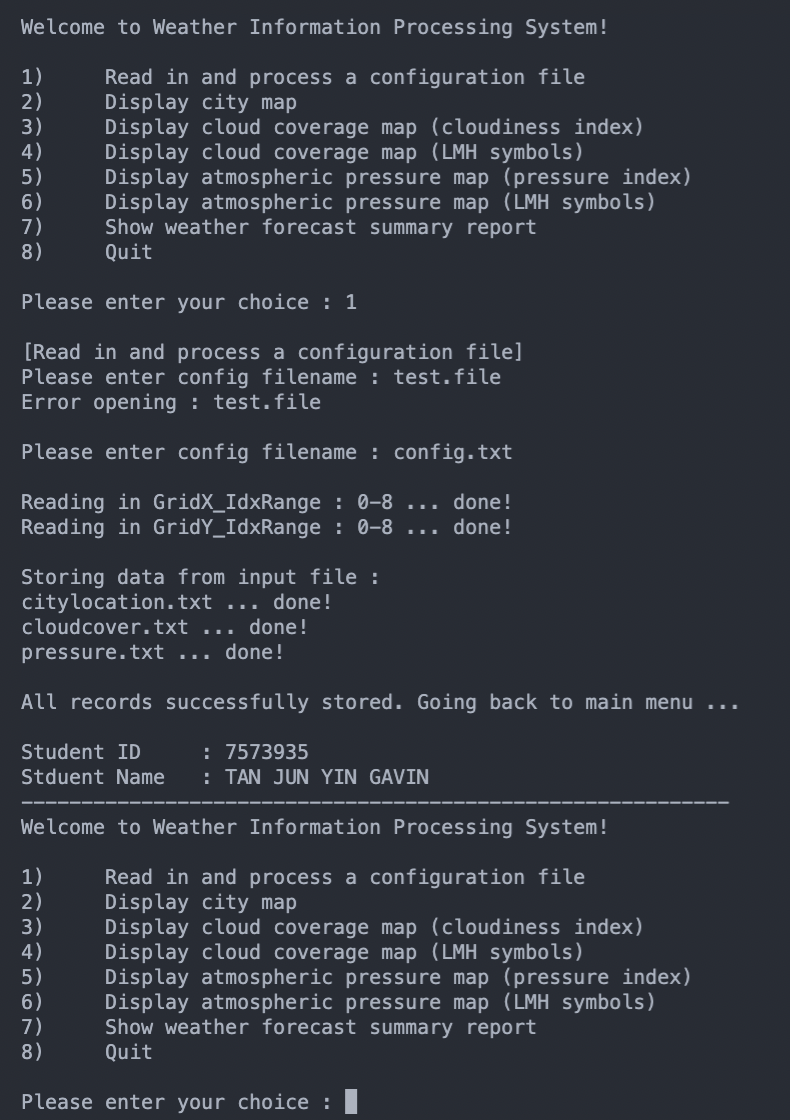
* A start-up dashboard to show the menu and some details (studentID + name)
* Individual modules/functions for each of the 7 menu options
  + To read and process a configuration file provided by the user through user input
  + Print out a “City Map”
  + Print out a “Cloud Coverage Map”
    - Can alternate between cloudiness index or LMH symbols
  + Print out an “Atmospheric Pressure Map”
    - Can alternate between cloudiness index or LMH symbols
  + Show weather forecast summary report
* Relevant error prompting and handling wherever needed

# **Program Design & Implementation**

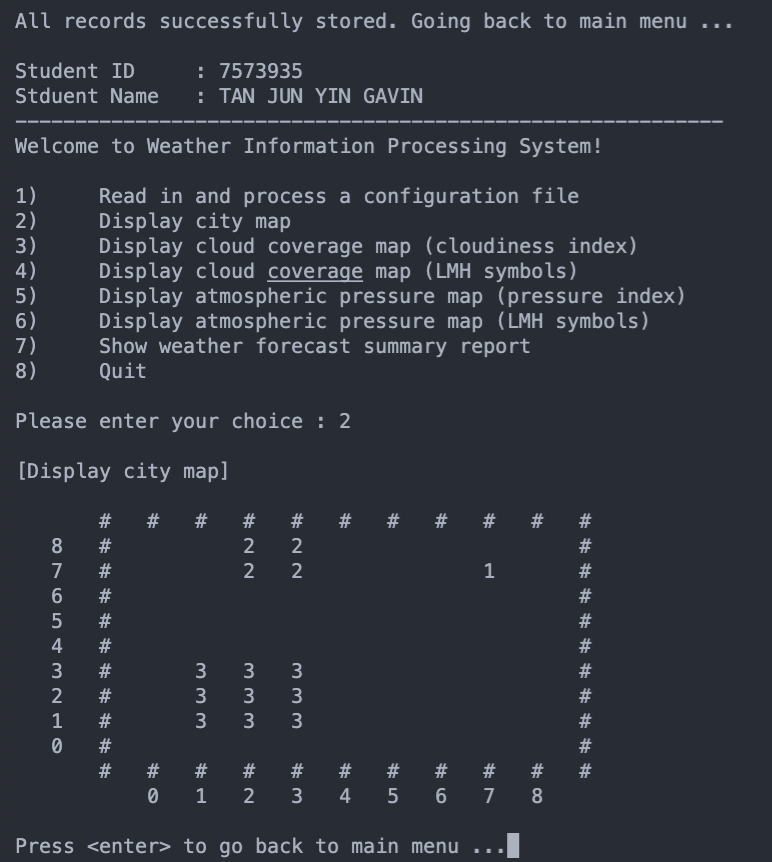
## **Program Flow**



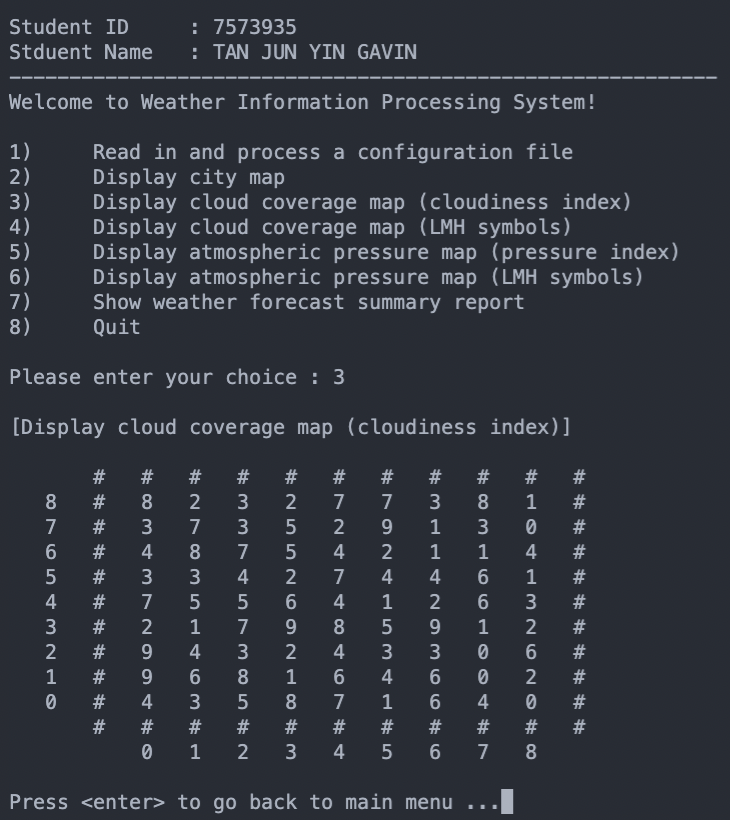
When we first run the .exe file, we will be greeted with a start up dashboard with the student ID and name followed by 8 menu options.



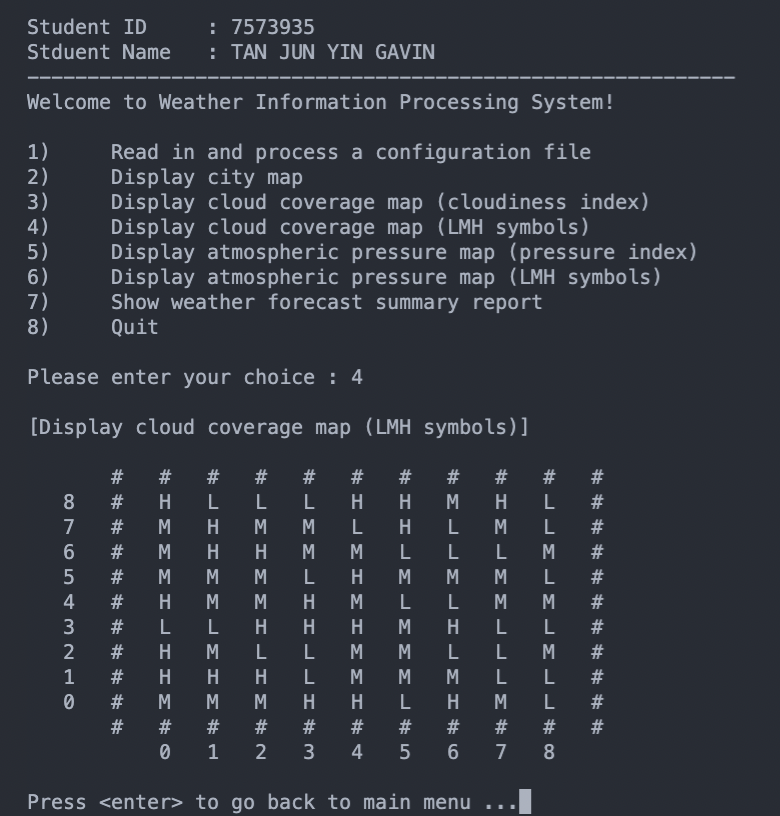
Firstly, we will read in and process the configuration file. As you can see above, I first tested with an invalid file to see what is the error prompt, followed by a valid file that will show that the records are successfully processed and stored.



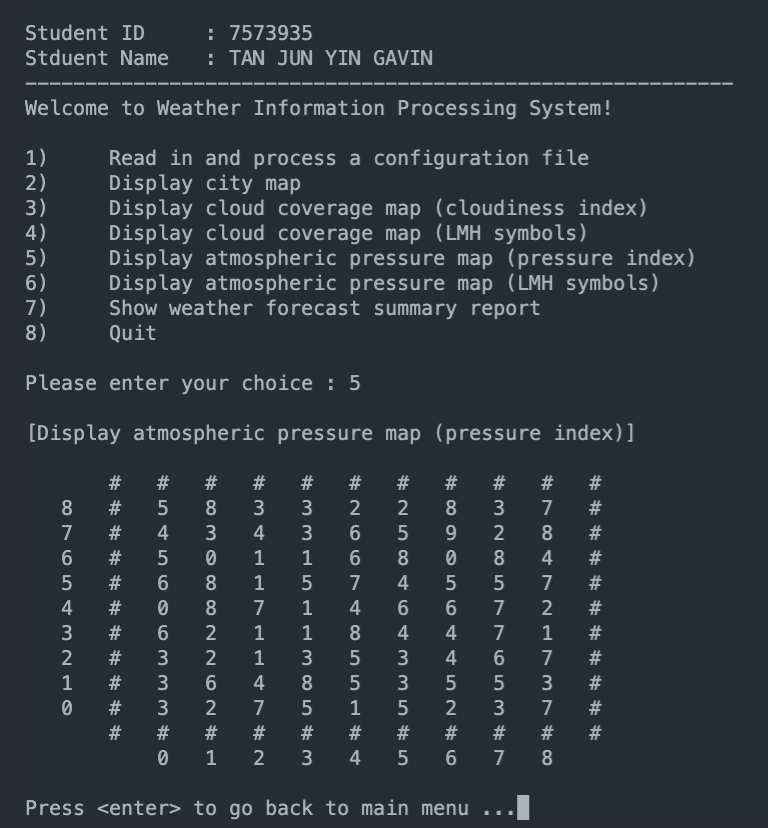
Next we will go through each of the menu options starting with “2” which is to display the city map based on the configuration files.



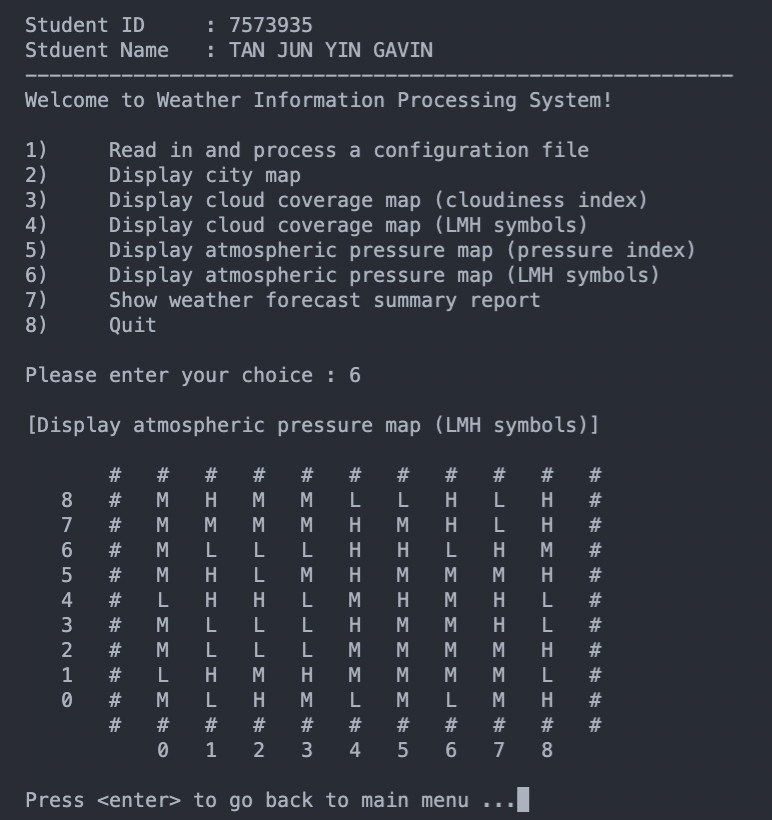
Then we will proceed to option “3” which is to display the cloud coverage map based on the cloudiness index.



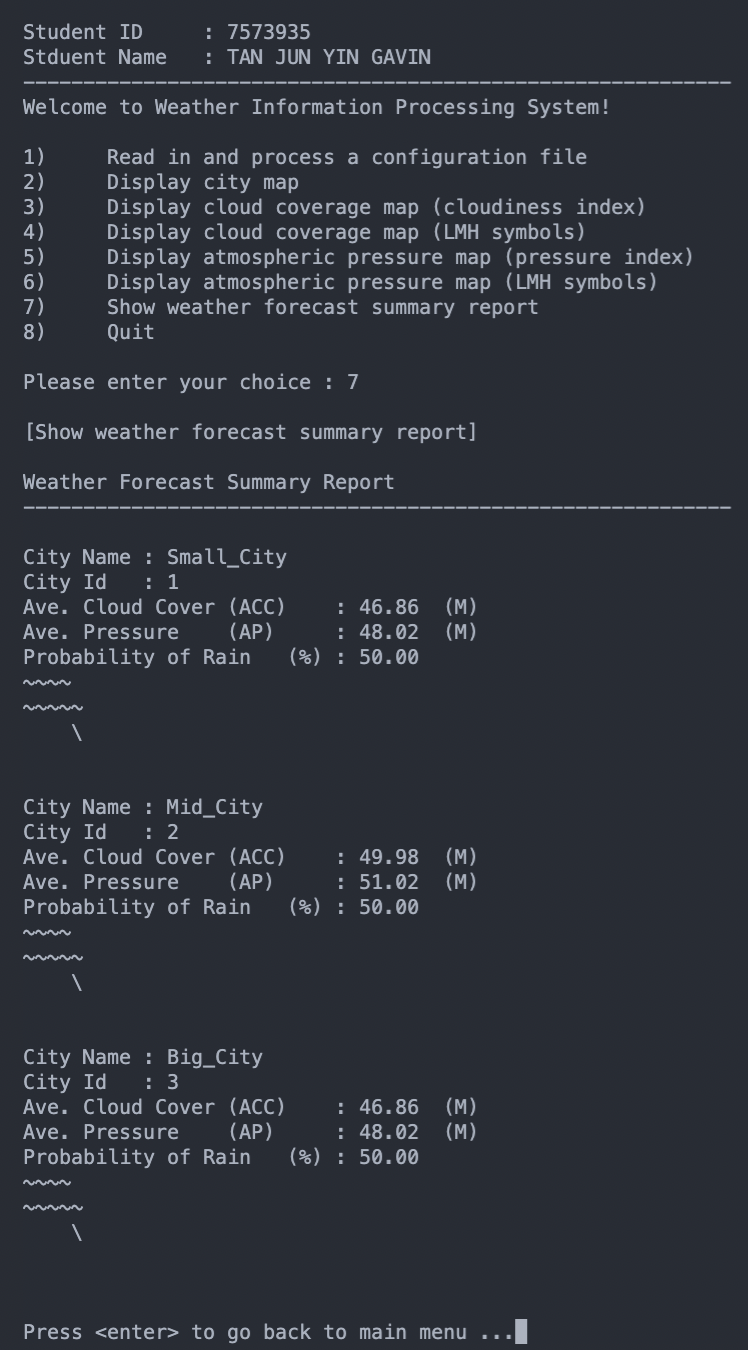
Then we will proceed to option “4” which is to display the cloud coverage map based on LMH symbols instead.



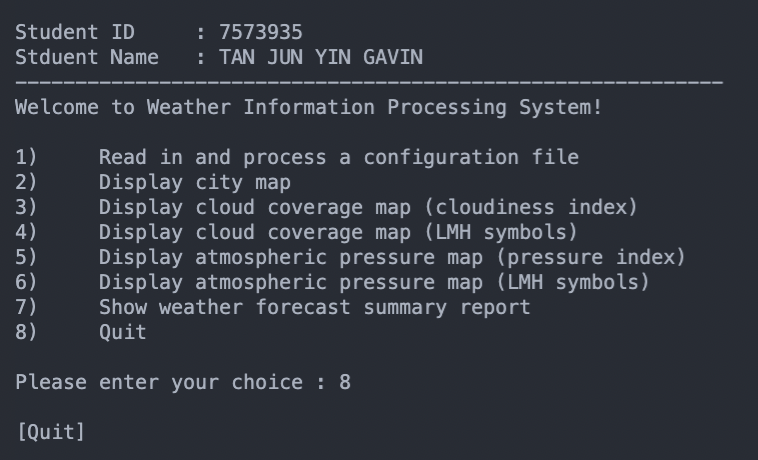
Then we will proceed to option “5” which is to display the atmospheric pressure map based on the pressure index.



Then we will proceed to option “6” which is to display the atmospheric pressure map based on LMH symbols instead.



Then for option “7”, it will show the weather forecast summary report



And last but not least, for option “8” it will quit the program.

## **Files and Functions**

### Readfile

* **void** processConfigFile()
  + Asks the user for a config filename
    - It will then attempt to open the file
      * if successful it will return the filename as a string
      * if not, it will return an error saying that there was an error opening the file/path which the user has input. It will then re-prompt the user for another file/path.
  + Then it will open the config file(filename) and read its contents
    - It will first check if the current line is an ID range for the grid maps
      * If so it will call readIdxRange to configure the grid map
    - it will then check if the grids have been configured (X and Y axis)
    - Then it will check if the current line contains “.txt”
      * If so, it will read the text file by calling readTextFile
  + Once the file has been read, it will close the file
* **void** readIdxRange(string readConfigLine, size\_t found\_IdxRange, int &count)
  + The function will store the min and max X and Y axis values into mapSize
* **void** readTextFile(string textFileName)
  + The function will attempt to read a text file
    - If there is no such file, it will prompt any error
  + Once the file is open, it will check if the file contains a “-” and “,”
    - If so, it will split up and tokenize the entire line into data and coordinates, storing them into the respective dataVector and coordinatesVector. After which, it will initialise the vector members
* **void** initialiseVectorMembers(string textFileName, const vector <string> &dataVector, const vector <string> &coordinatesVector)
  + Initilizes vectors with data from the textFile
* **vector<string>** tokenizeString(string str, string delimiter)
  + The function attempts to tokenize the string by splitting it up based on the given delimiter

### Structs

* The structs header and cpp files are meant to set standardized structures for each dataset we need to manage.
* We have 4 structs:
  + XYrange
    - Has 4 int values (minX, minY, maxX, maxY)
  + CityLocation
    - Has 3 int values (Xcoordinate, Ycoordinate, cityId)
    - Has 1 string (cityName)
  + CloudCover
    - Has 3 int values (Xcoordinate, Ycoordinate, cloudCoverValue)
  + AtmosphericPressure
    - Has 3 int values (Xcoordinate, Ycoordinate, atmosphericPressureValue)

### Map

* **Void** displayMap(int userChoiceNumber)
  + The function will display the map/grid accordingly to the values found in the config files
  + Firstly, it will calculate the number of rows and columns and then allocate the values into a 2D array which will then be initialized
    - Then, it will start to print out the map starting with an outer for loop for the Y axis and an inner for loop for the X axis.
    - Each loop operation will check if the current pointer is supposed to print out a value from the 2D array or is it meant to be blank or if it’s the border
* **Void** print2DArrayValue(int value, int userChoiceNumber)
  + The function will print the 2D Array Value based on which option the user has chosen
    - For options 4 and 6, it will display the LMH symbols
    - For options 3 and 5, it will divide the value by 10 and print out the first digit only
    - For all other options, it will just display the original value
* **Void** printData(string data)
  + This function is used to standardize the displaying of values using setw to set the width
* **int** calculateCoordinate(int index, int minIdxrange)
  + This function calculates the current pointer's coordinate based on the index and map’s minimum X and Y axis.
* **char** getLMH\_Symbol(int value)
  + This function is used to convert the numeric values into LMH symbols
    - For values between 0 and 35, it will print “L”
    - For values between 35 and 65, it will print “M”
    - For all other values, it will print “H”

### Summary Report

* In this file, we will first override the cityInformation::toString() function and format the output based on the assignment requirements
* **void** displayWeatherForecastSummaryReport()
  + This function will display the weather forecast summary report
  + It will first loop through the cityLocation vector and then store each cityId into a hashmap
  + Then we will iterate through the hashmap, extracting the information we need and storing it into cityInformation
  + Afterward, we will iterate through the cityInformation vector to initialize the grid’s X and Y axis as well as store the sum and average of each city
  + Lastly, we will iterate through the cityInformation vector one last time to print out the map using the override .toString() function
* **void** store\_Original\_XYrange(CityInformation &city, const int index)
  + This function will help store the original X and Y axis range for each city
* **void** store\_Final\_XYrange(CityInformation &city)
  + This function will help find the final range of the X and Y axis for each city
* **void** findRespectiveData(int &minX, int &maxX, int &minY, int &maxY, int &count, string &cityName, const int index)
  + This function is used to find the respective min/max values for the X and Y axis for each city
* **int** getMinValue(int a, int b)
  + This function compares 2 values and returns the smaller value
* **int** getMaxValue(int a, int b)
  + This function compares 2 values and returns the larger value
* **void** storeSumAndAverage(CityInformation &city)
  + This function is used to store the sum and average of the cloud cover and atmospheric pressure for each city
    - It will first get the min and max range for both the X and Y axis
    - Then it will have an outer for loop for the row and an inner for loop for the column. Of which, it will accumulate the cloud cover and atmospheric pressure values based on the current location, using the row and col indexes as the pointer
    - Then it will also calculate the average for the cloud cover and atmospheric pressure
    - Lastly it will store the values back into the city variable
* **void** getCloudCoverValue(int rowIndex, int colIndex)
  + This function is used to get the cloud cover value based on the rowIndex and colIndex, acting as a pointer on the “grid”
    - It will use the coordinates of the pointer and check if there is a value in the cloudCovers vector. If so, it will return the value and if not, it will continue to loop through the vector
* **void** getAtmosphericPressureValue(int rowIndex, int colIndex)
  + This function is used to get the atmospheric pressure value based on the rowIndex and colIndex, acting as a pointer on the “grid”
    - It will use the coordinates of the pointer and check if there is a value in the atmosphericPressure vector. If so, it will return the value and if not, it will continue to loop through the vector
* **double** getTwoSigFigValue(double value)
  + This helper function is used to calculate and return the double variable, rounded off into 2 significant figures
* **void** storeRainProbabilityAndGraphics(CityInformation &city)
  + This function is used to store and display the rain probability using the LMH symbols
* **string** createGraphics(int firstLine, int secondLine, int thirdLine)
  + This function is used to display the rain probability according to the assignment requirements
* **void** appendCharacters(string &str, int noOfTimes, char character, bool isAddnewline)
  + This function is used in tandem with createGraphics() to append characters, into each line

### 2D Array

* **int** allocate(int rows, int cols)
  + This function is used to instantiate and return a p2DArray based on the number of rows and columns
* **void** initialize(int \*\*p2DArray, int rows, int cols)
  + This function is used to initialize the 2D Array
* **void** assignValues(int \*\*p2DArray, int userChoiceNumber)
  + This function is used to assign values to the p2DArray based on what option the user input in the menu
    - If they were to input “2”, which is to display the city map, it will iterate through the cityLocations vector, calculate the respective X and Y coordinates, and then assign the cityId to the respective X and Y coordinates
    - If they were to input “3” or “4”, which is to display the cloud coverage map, it will iterate through the cloudCovers vector, calculate the respective X and Y coordinates, and then assign the cloud cover value to the respective X and Y coordinates
    - If they were to input any other option like “5” or “6”, which is to display the atmospheric pressure map, it will iterate through the atmosphericPressures vector, calculate the respective X and Y coordinates, and then assign the atmospheric pressure value to the respective X and Y coordinates
* **int** calculateCoordinate(int coordinate, int minIdxrange)
  + This helper function is used to calculate the coordinates based on the minIdxrange. It is mainly used to check if the IdxRange is not equals to 0, for example whereby the minimum X coordinates start from -1 or 1, it will adjust the numbers accordingly before inserting it into the p2DArray
* **void** clear(int \*\*p2DArray, int rows, int cols)
  + This function is used to clear and delete the p2DArray after it is done being used to help save space and prevent any memory leaks

# **Reflections**

## **Difficulties Faced**

A big part of the difficulties faced would be deciding what data structure to use so that the implementation can be done in the easiest way possible. I struggled between whether or not to use vectors, arrays, or lists.

Another difficulty faced would be the printing out of information, such as the map because you have to use a pointer as a reference to check if the X and Y coordinate you are currently on, has a value to print or not. You also need to be mindful and check if the current pointer is at the border or not.

## **What Could Have Been Done Better?**

I believe one area that could still be improved would be to write cleaner code, utilizing better alternatives to help reduce space and time complexity, especially when it comes to sorting and looping. I have a few nested for loops that could have been improved using alternate solutions.

## **What Have You Learned?**

1 key skill/knowledge which I have learned through this assignment is to modulize and break down chunks of code that are being repeated, into reusable functions, thus creating helper functions.