CSCI251 Assignment 1 Program Document

# Program Requirements:

A brief explanation of this assignment is as such, this program will firstly read a config file to obtain the grid indices followed by 3 file names. The 3 filenames are pointing to files that will contain respectively the city location, cloudiness index values and pressure index values. The 3 files will then be respectively read a line at a time to obtain the grid’s X , Y coordinates and the value to be stored within. The values will then be stored in a 3D array of which the implementation will be further elaborated. The final requirement is to calculate the rain probability of the cites by cross referencing the cloud and pressure values of the respective indices. Additional features were added as recommended in the assignment document.

The first main function is to prompt a user to input a file name that relates to the config settings, as for the current test case given for this assignment, the file is called “config.txt”. All grid values found are preceded with a GridX\_IdxRange or GridY\_IdxRange for consistency, it will then be further assumed that the next 3 lines of text in the config file ending with “.txt” will be the city location values, cloud cover values and pressure values respectively. My program will then take in the file names and proceed to open and store the values respective to their X, Y values into a dynamic 3D array. The array has to be dynamic as the grid indices can only be known during runtime.

The second function will be to display the values stored in a transposed grid formation representing a map which essentially is a 2D array relative to the data type (city location/cloud/pressure). The map has to be transposed as most conceptualizations of 2D arrays have the 0,0 coordinates at the top left of the diagram whilst this program displays all its values with the 0,0 coordinates at the bottom left of the diagram. The values represented in the map for cloud and pressure are int casted to the value divided by 10, as the grid only represents the value in the ten’s position. There is also a sub function to display the values in (L, M, H) symbols. As for city values, the City ID is only displayed while the areas not occupied by any city will be blank.

The final function will be to compute the rain probability and print a rain summary report.

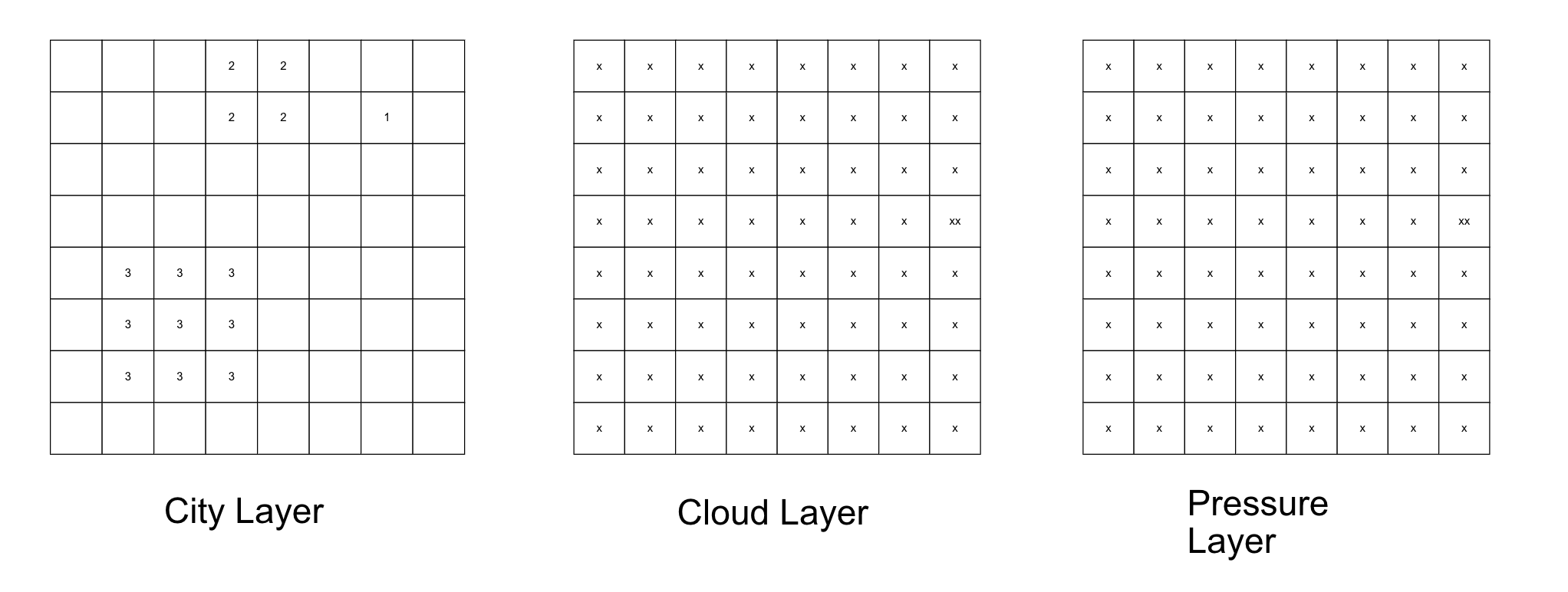
This can be achieved by cross referencing the cloud and pressure values on the city’s location whist +1 of its surroundings and extrapolating the rain probability with the table given in the assignment pdf to derive an accurate probability for each city in the grid.

As for the additional functions added, they will be further explained in this document.

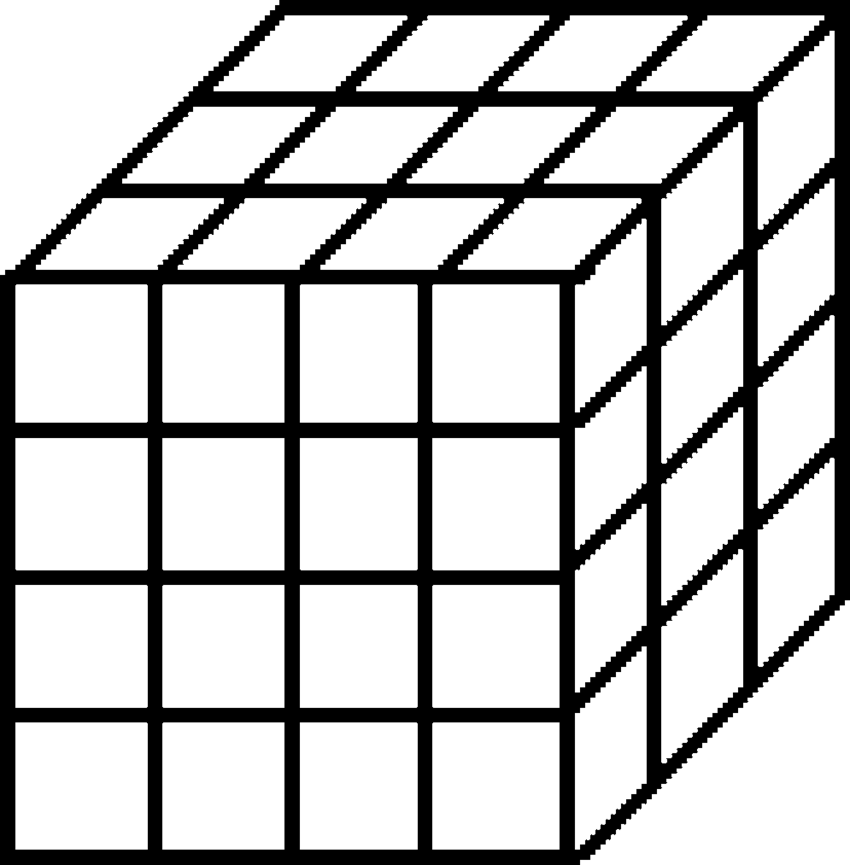
# Program Design:

To store all the different layers of mapped values for this program, I have decided to use a 3D dynamic array and pointer arithmetic to traverse the array.

2D blow out view of the 3 “layers”:



3D conceptualization of the array model:



Pressure layer

Cloud layer

City layer

This will allow for efficient cross-referencing data on the same grid without the use of classes, a simple “layer change” would be sufficient to access the different value types on the same coordinates.

This method of storing the 3 layers of mapped values on a 2D grid/map is essentially the crux of the whole program. This also allows of efficiency for certain cases where during traversal and a different “layer” value is needed on the same coordinates. For example, I am currently traversing the array on the city level and require the cloud layer value of same location as the CityID( this was a use case for calculating rain probability ), I would simply change the first index layer ‘x’ to 1 (3Darray[x][y][z]) to access the cloud value of the respective coordinates.

# Summary on each program module:

## Read in and process a config file

This module reads in a config file that matches the name of the user’s input. For testing the file that was used was “config.txt”. The file should consist of 2 Grid values of X and Y respectively. The format of the value should follow “GridX\_IdxRange=0-8” or “GridY\_IdxRange=0-8” where the value preceded by the “– “will be used for the X or Y index respectively. Upon extracting the X and Y indices, the module will then create a 3D dynamic array as such 3DArray[3][X][Y], the first index is always 3 due to the fact it’s an aggregate of 3 2D maps. Additionally, all values in the new 3D array will be normalized to 0 to avoid garbage values being displayed during the other modules.





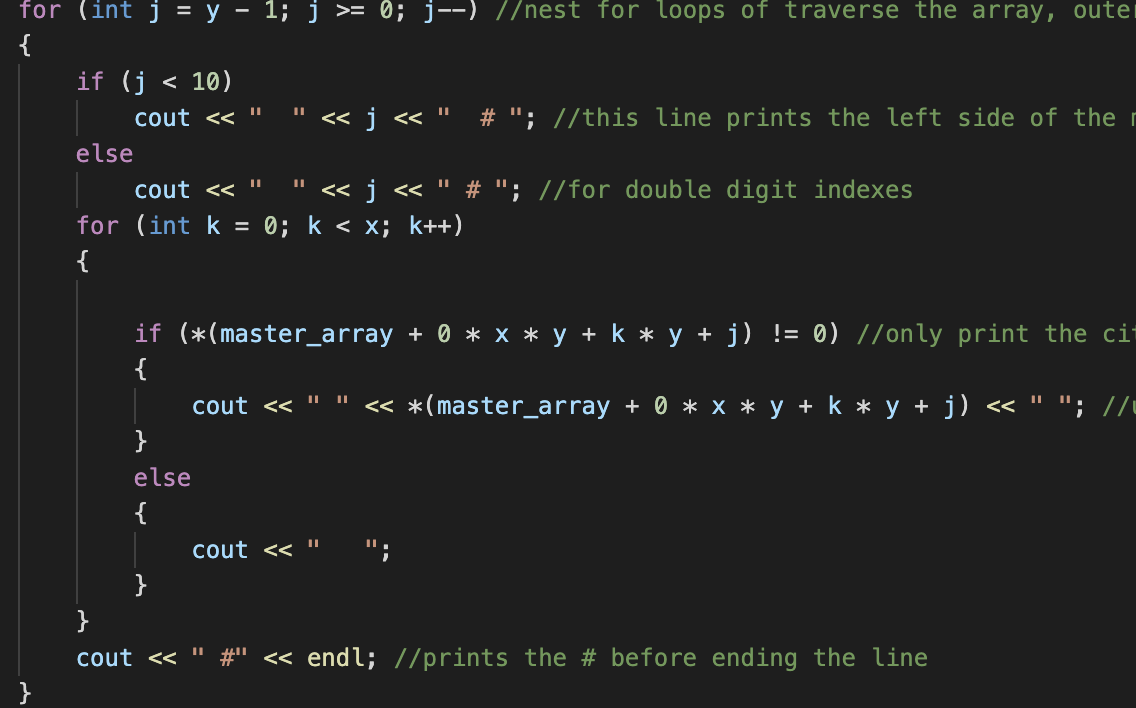
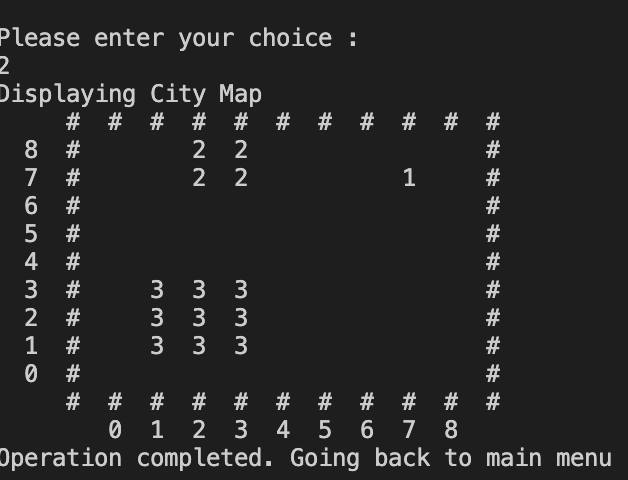
The module will then further read line by line in the config file and search for a text match where a string has a text that ends with a “.txt”. what is preceded by the “.txt” is omitted as the test cases that will be provided by the lecturer will be this specific order [City, Cloud, Pressure] regardless of file name so long as it is a text file. Upon hitting a “.txt” match, the module will further call another read file function and read line by line till end of file. At every line of .txt files will have an X-ordinate, Y-ordinate and Value in this format ([1, 1]-3-Big\_City) for city file or ([1, 1]-80) for cloud and pressure file. As the format has the x and y coordinated, it will directly map the values into the 2D array of the respective data type (city, cloud, pressure). Additionally, a global string vector will be created to store only the unique city names that is found in the city file.

## Display city map

This module displays the location of each city on the map by traversing the array and displaying the output in the terminal if there is a cityID value present or a blank space if there is none. The method of traversal is using pointer arithmetic and nested for loops.

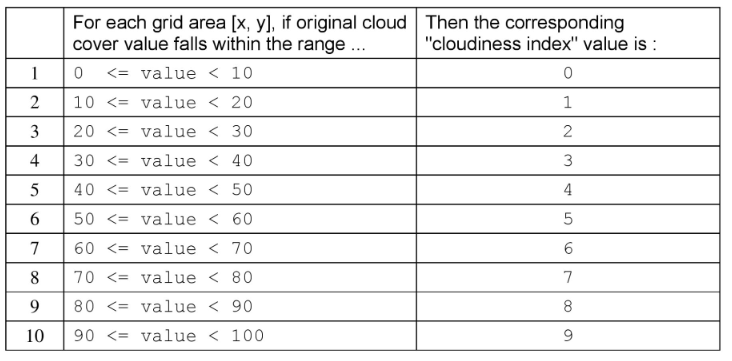
Additionally, to print the top and the bottom of the map, a for loop is needed to be able to match the contents of the map as the X and Y indices are only known during run time.

The entire array will also need to be transposed as traditionally most 2D array outputs have the 0,0 coordinates at the top-left of the display. But for this program, the 0,0 coordinates are the bottom-left of the display. Therefore, the map needs to be transposed by having the Y index decremented to 0 in the outer loop and X index incremented to X from 0 in the inner loop as such.

## Display cloud coverage map(index/LMH) & pressure coverage map(index/LMH)

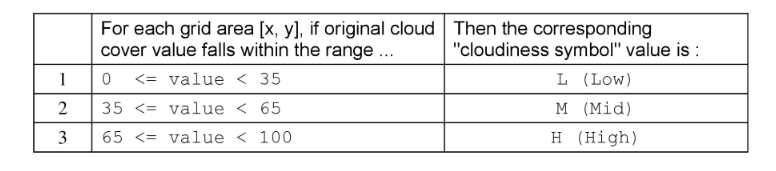
This module performs similarly to display city map module, it traverses the 3D array on the cloud/pressure layer and displays a value from 0-9 relative to the stored cloud/pressure value. The 0-9 value is chosen via this table as shown below:



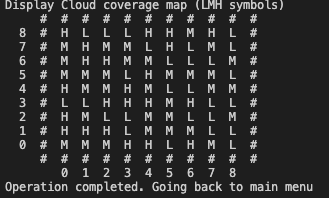
Once the conversion is complete, the corresponding index will be displayed on the map.

Same as the city map, a function will be called to print the top and bottom of the map relative to the gird sizes as such.



As for the LMH symbols, the values are chosen based on this table: 

So instead of converting a value of 95 to 9, it will be converted to a “H” instead which will lead to an output as such



## Show Weather Forecast summary report

This module computes the Average Cloud Cover (ACC), Average Pressure (AP) and based on the calculated values it displays a forecast report for each respective city.

Firstly, it will obtain the ACC and AP in 2 stages, it will traverse the array and get all the coordinates where CityID equals to 1(Small city) for example. It will then sum all the cloud cover values and pressure values for grids with the cityID of 1. Next the program will traverse to each individual coordinate and find a cityID of 1 again, but instead of adding the cloud/pressure values, it will get the surrounding cloud/pressure values by following this guideline:

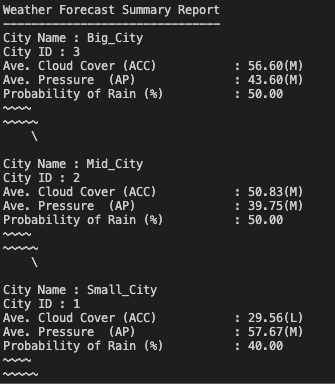


It will append all the surrounding coordinates into a vector and will be further processed to remove all the duplicate coordinates as cities with an area greater than 1 unit square will have overlapping surrounding coordinates.

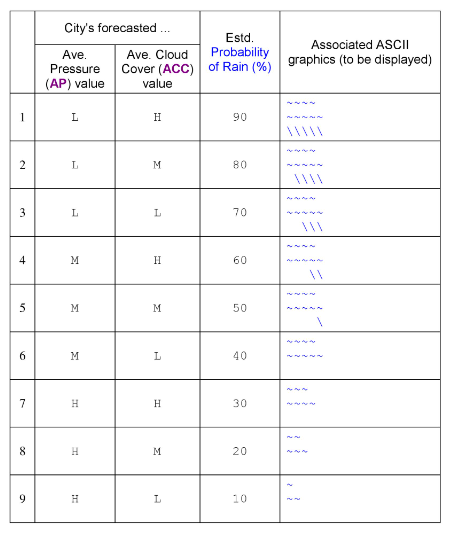
Once all the unique coordinates have been stored. The program will loop through each vector item and dereference the coordinates storing the cloud/pressure value and appending it to the sum.

It is quite apparent that this is an inefficient way to get the surrounding data as there are plenty of overlaps and simply by getting the perimeter of the city could be a possible more efficient way to get the surrounding cloud/pressure values. But this was implemented this as an additional feature due to the fact that it will be able to get the surrounding values for cities that are not in the shape of a square or rectangle. It could also get the surrounding values of cities in a diagonal line shape.

Finally, once the ACC and AP has been summed and averaged. The program will proceed to do the display portion of the program:



The LMH symbol after the ACC and AP values are exactly the same as the LMH conversion as for cloud and pressure maps. Once the LMH values are computed, the rain probability can be calculated by following this table:



Once calculated, the program will display the probability value and corresponding ASCII art.

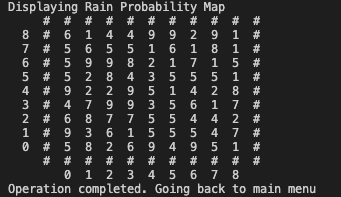
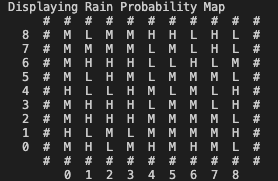
## Rain probability map(index/LMH) – Additional Feature

This module gets the rain probability value of each grid in the map. It works similar to the way the program computes for rain probability for the summary report, only it does so regardless whether a city is present or not.

It will traverse the 3D array and deference the 2 layers simultaneously to get the cloud and pressure values and referring to the LMH guide as shown previously. The values will be assigned to the corresponding LMH and with the values a rain probability will be assigned.

As probability of rain are in the 10’s percentage, therefore the program will only display the digit in the 10’s position which will not lead to a lost data. As for the LMH version of the display, L would be assigned to any values < 4, M would be < 7 and H would be everything above.

Here is a sample output with the given config file.

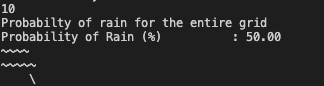
 

## Rain probability of the entire map – Additional Feature

This module would get the sum of the entire ACC and AP of the whole map. Then proceed to calculate the probability of rain with the LMH values of its ACC and AP. It traverses the entire 3D array and sums the cloud and pressure values. Then proceeds to compute the average by dividing the sum with X multiplied by Y as that represents the total number of grids in the map.

Once ACC and AP have been calculated, referencing the probability table shown previously, the corresponding rain probability and ASCII art will be displayed.

Here is the sample output for the given config file.



# Reflections

This program assumes that the config file does not mis-order the city, cloud and pressure txt files as doing so will break the program, additionally there are limitations on output via the terminal, therefore grid indices greater than 99 would also cause inaccuracies in the program. The assumption is also made that there would not be a city ID greater than 9 and cityID = 0 though it would be easily rectified if the use case is required. The final assumption is that all city, cloud and pressure files follow the correct format and the coordinates are not out of bounds causing memory issues in the program during the reading and displaying modules.

Challenges that I initially faced were, difficulty in transposing the array display to match the expected output as the 0,0 starts from the bottom of the display. Additionally, during the early stages of this assignment, I did not normalize all the 3D array data to 0 which caused random garbage output leading to tons of confusion. This sent me on a wild goose chase to do redundant debugging on functions/modules that were working correctly. Additionally, I faced a challenge when I was trying to navigate and use pointer arithmetic for 3D array traversal as pointers were always a weak point for me. It also didn’t help that this is the first module right after enrollment and I am pretty rusty when it comes to programming especially in C++ where the last time I coded in to for a school assignment was in 2017.

An area of which I could have improved on this program is on code reusability, there were many over-laps within the program like array traversal. Additionally, the implementation of structs for the mapped data would also be an alternative way to store and organize the data given in this assignment but seeing as classes were not allowed, I decided to take up the extra challenge of not using structs as well.

If I had more time, I would think adding a feature to find the probability of rain given 4 coordinates in a shape of a rectangle or a square would me a good use case for this program.

Despite all the challenges faced, during this assignment I have developed a mindset to be an independent learner. Whilst developing a skillset in information filtering such as using google search and stack overflow efficiently to find specific algorithms and code snippets that allowed me to solve the bugs and errors, I was facing regardless of how relatable the use case was. Finally, it rekindled my self-learning attitude and seek out content on topics that I had poor conceptual knowledge on without waiting to be spoon fed solutions.