Student Name : Yang Yiling

SIMID : 10267626

UOW ID : 9090307

Subject : CSCI251 - Advanced Programming

Tutorial Class : T06F

TABLE OF CONTENT:

[Assignment requirement/ Introduction:](#_n7o6td2y1dit) 1

[Option 1:](#_eehtpmggg6ka) 3

[Option 2-6:](#_pfrbb6jz3cb3) 5

[Option 2:](#_om44qm19c8vf) 7

[Option 3 and 5:](#_gypvqtij1t5m) 8

[Option 4 and 6:](#_hdj3vrnkh9m6) 10

[Option 7:](#_ht5b1i9ofe0x) 12

[Functions Used](#_195tpwq8tan2) 14

[Additional Features:](#_k74c0uqdcwyh) 18

# Assignment requirement/ Introduction:

The assignment requires me to develop a Weather Information Processing System (WIPS) using procedural programming in C++, requiring the use of control structures, arrays, dynamic memory allocation, and input/output operations. It involves reading and processing a configuration file (`config.txt`) to set up program parameters and extracting weather data from input files (`citylocation.txt`, `cloudcover.txt`, `pressure.txt`). The program should display a city map as a grid with city IDs at their respective coordinates. Additionally, it should show cloud coverage and atmospheric pressure both as single-digit indices and using LMH symbols (Low, Medium, High). A weather forecast summary report must be generated, displaying city names, IDs, average cloud cover (ACC), average pressure (AP), probability of rain, and ASCII graphics representing the rain probability. The program should also include a main menu for users to choose options, such as displaying maps or generating reports. Below are the codes used for the assignment with the output when running in the Ubuntu terminal using g++ compiler.

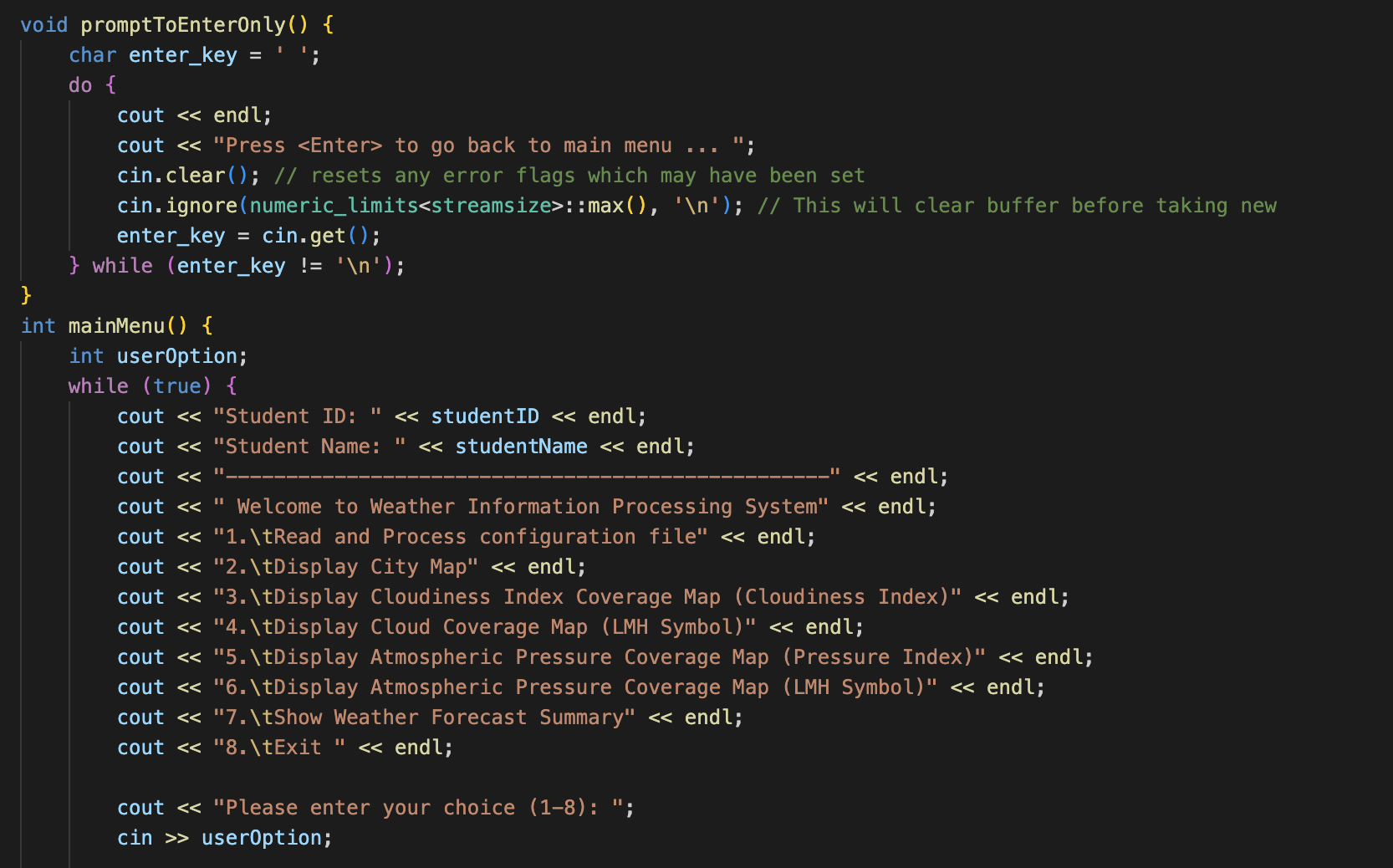
Compiling the file in Ubuntu:

g++ -std=c++17 FT\_T06F\_A1\_10267626\_YangYiling.cpp -o a1.app

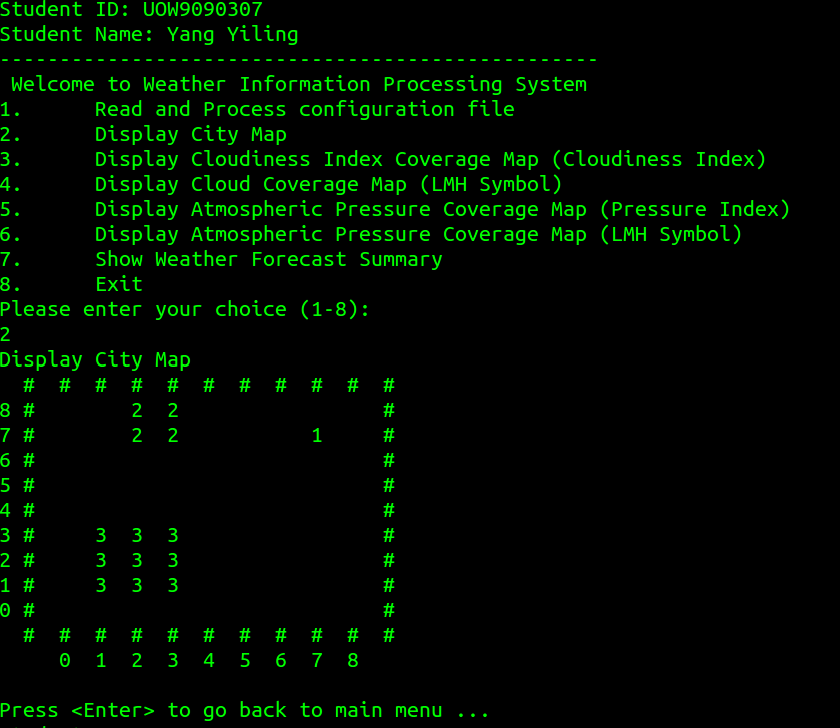
Run file in Ubuntu:

./ a1.app

Upon running the app file in Ubuntu, users will be directed to the mainMenu where they are required to select option 1-8. This will allow for user interaction, based on the selection of the user, it will direct them to different output screens. Users are only allowed to select option 1-8, when selecting outside the range it will show that it is an invalid choice.

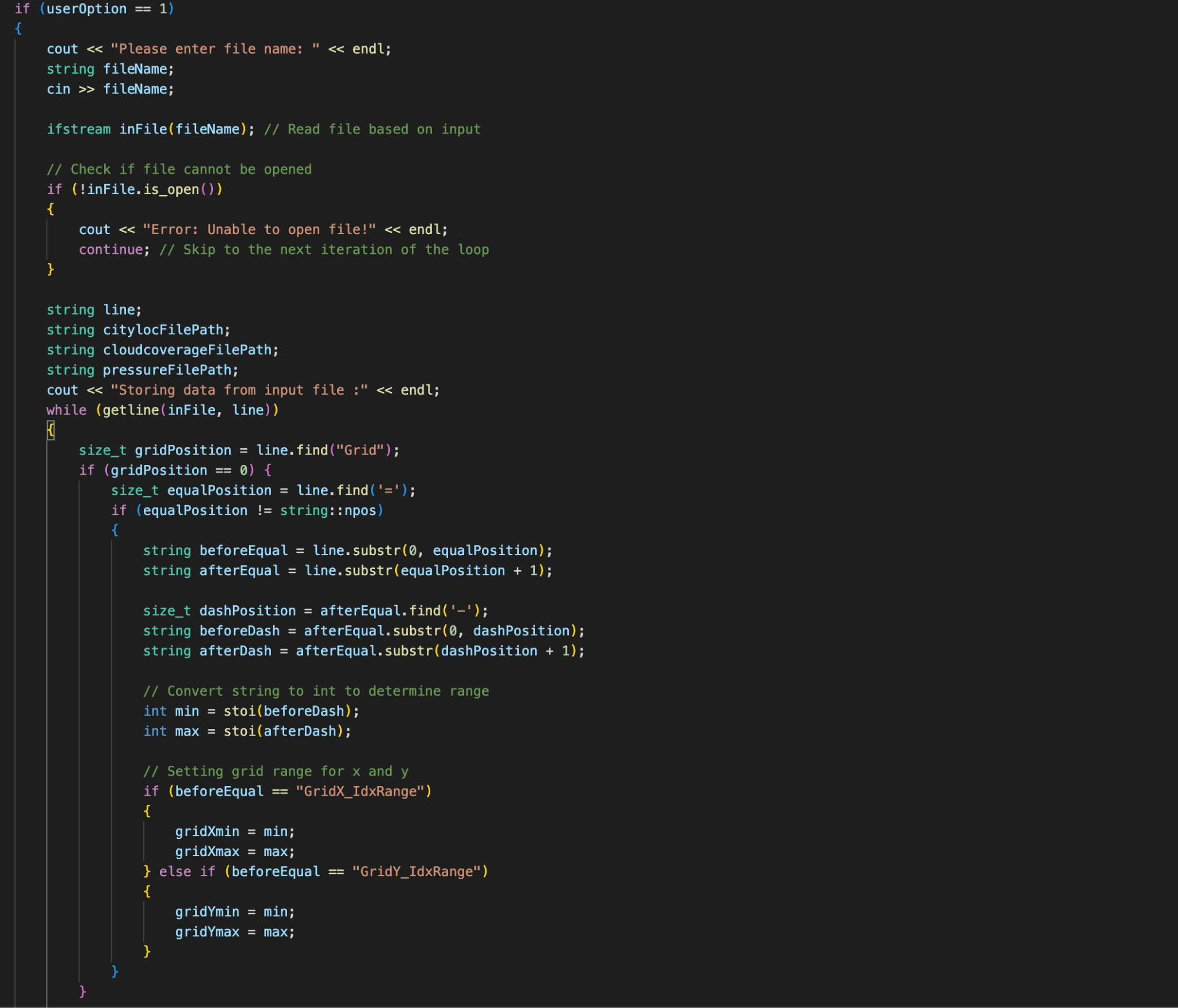


Main Menu display in Ubuntu:

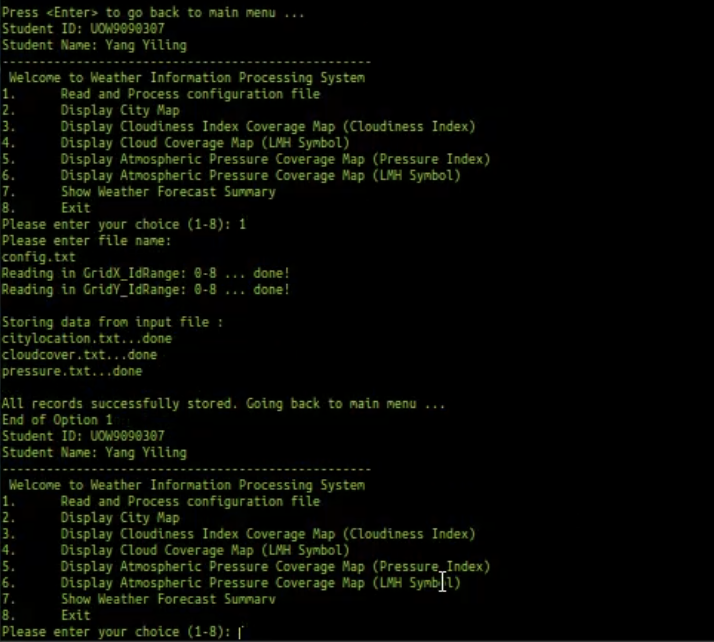


# Option 1:

When users enter Option 1, they are required to enter a file (config.txt)t. After running the config.txt file, it will display an output of the grid range for X-axis and Y-axis. It will also check if the citylocation.txt , cloudcoverage.txt and pressure.txt files are being passed through successfully. By using find( ‘=’) to obtain the range of minimum x and maximum x as well as for the y axis. After loading option 1, it will automatically bring the user back to the mainMenu. promptToEnterOnly is only used in option 2-7 where users are required to key <enter/return> key to direct the screen back to the main menu.

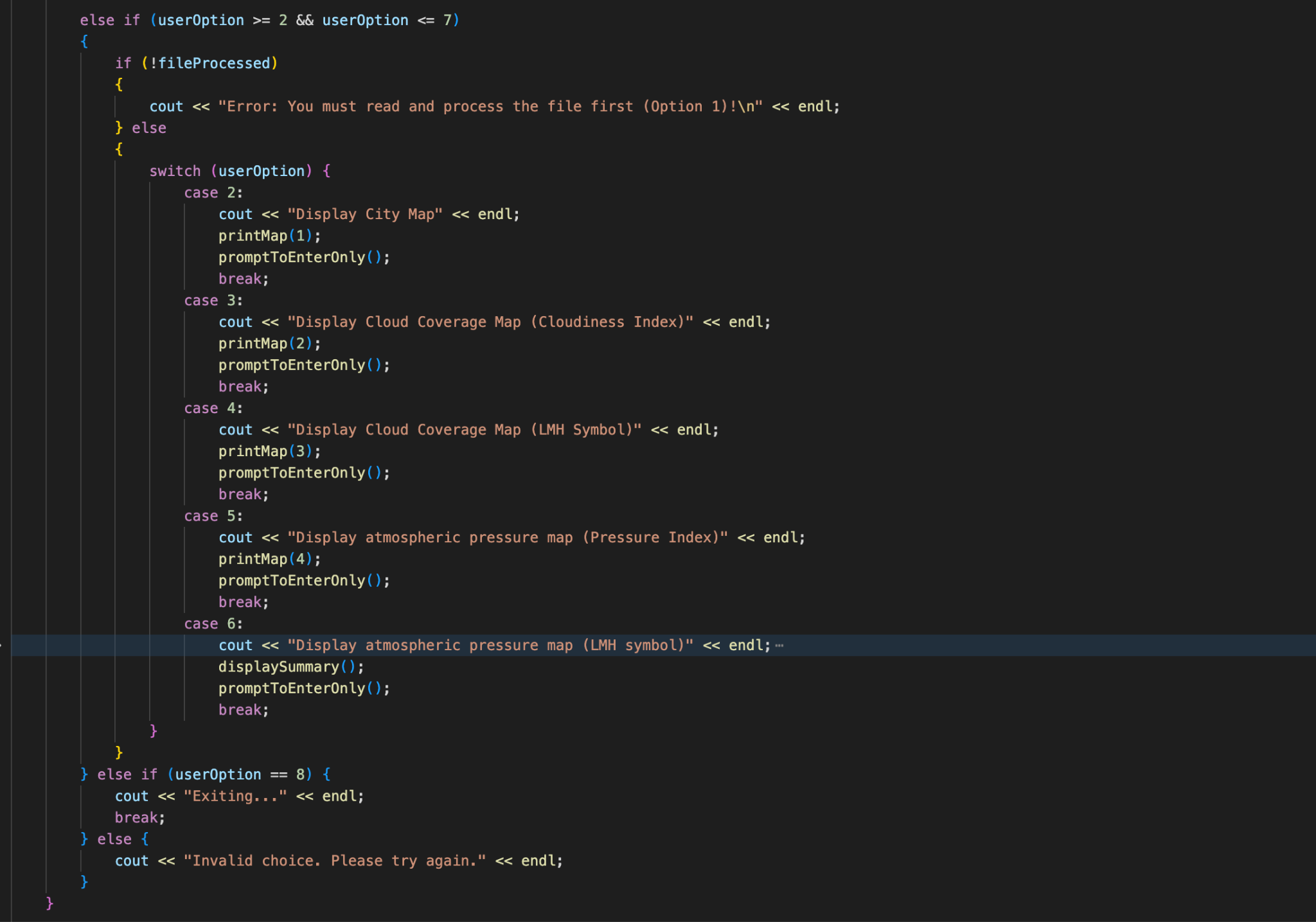
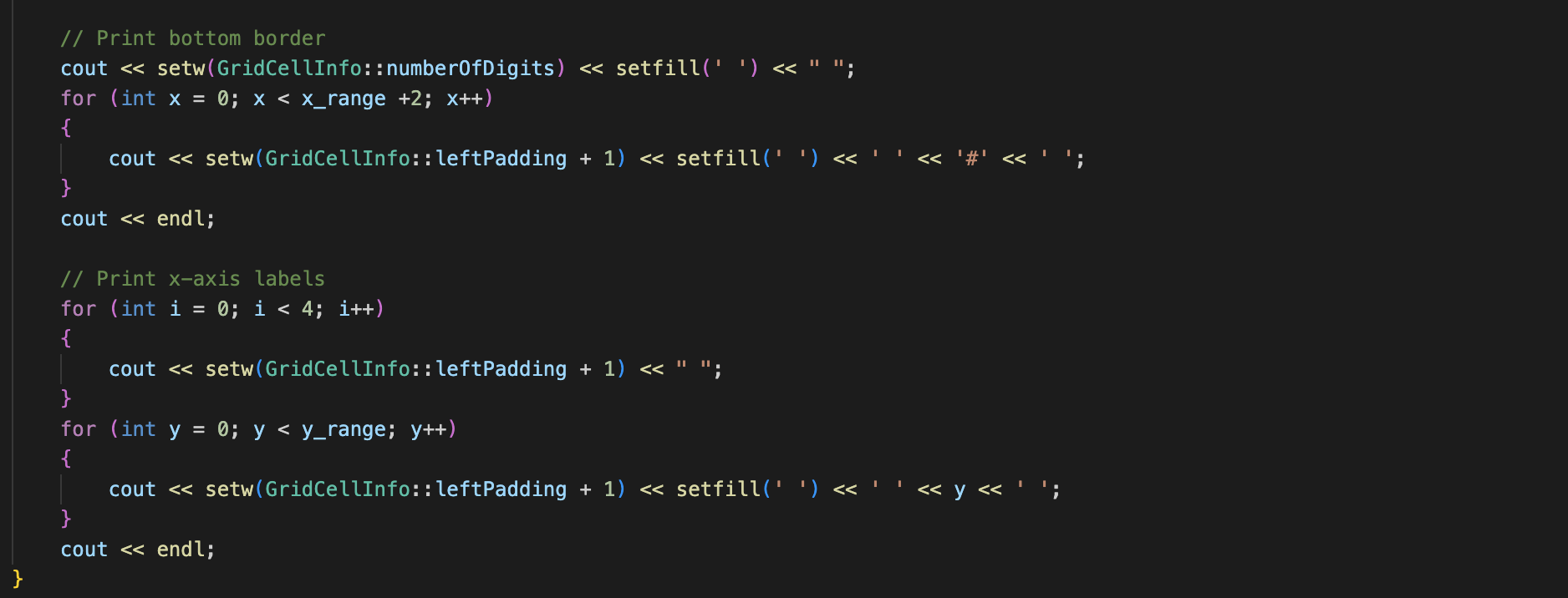
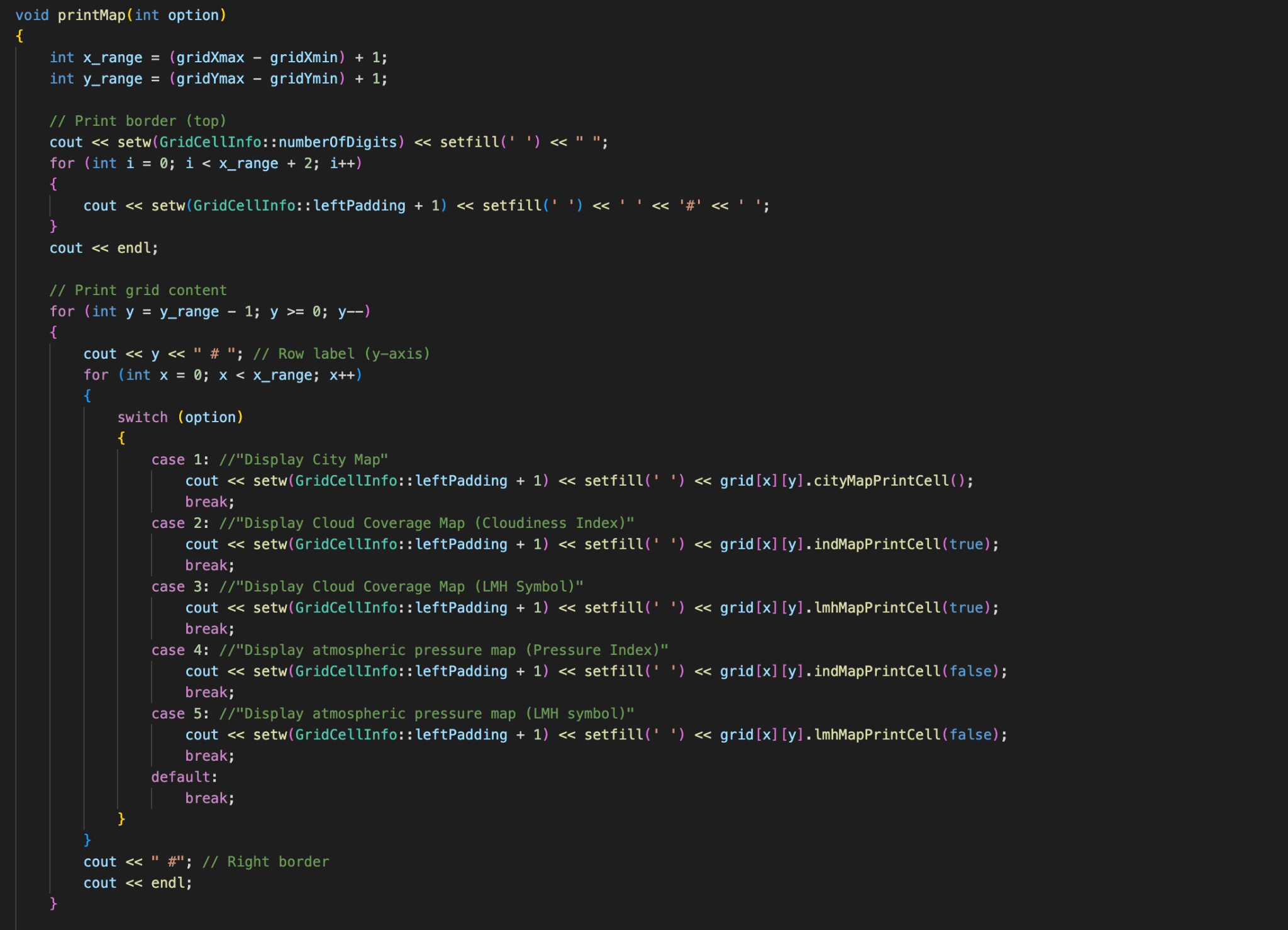


Output on Ubuntu:



# 

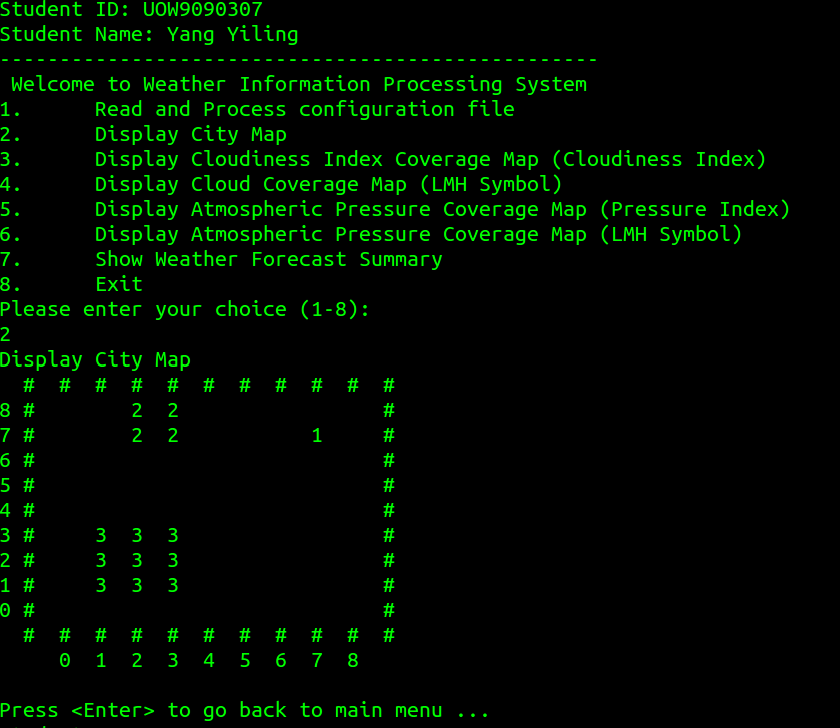
# Option 2-6:

Option 2 to option 6 will print the grid based on the various text files that were passed through. The grid will be plot based on the x and y range. To select option 2-6 users are required to enter option 1 inorder to retrieve the text file. If option 2-6 is selected before option 1, it will generate an error output stating it is required to process a file in option 1 first. The file will be passed through the printMap function for all the options 2-6, and are stored as integer option 1-5, where 1 is the city map display and 2 as the cloud coverage map etc. For each option user will be required to hit <enter> to return back to the main menu. The printMap function generates and displays a grid-based map based on the selected option, such as city maps, cloud coverage, or atmospheric pressure. It calculates the grid's dimensions, prints a top border, and iterates through each cell to display the appropriate content (e.g., city IDs, numerical indices, or LMH symbols) with consistent padding for alignment. After printing the grid, it adds a bottom border and labels the x-axis for clarity. 

# Option 2:

Option 2 will generate an output of the city map, showing the placement of the cities by city ID that was being abstracted from the citylocation.txt file

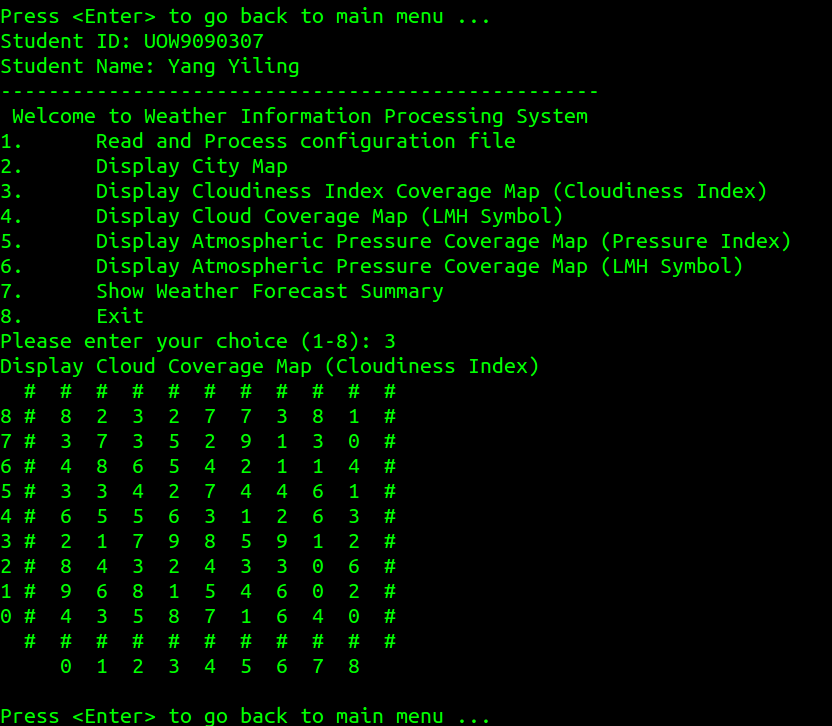
Output on Ubuntu:

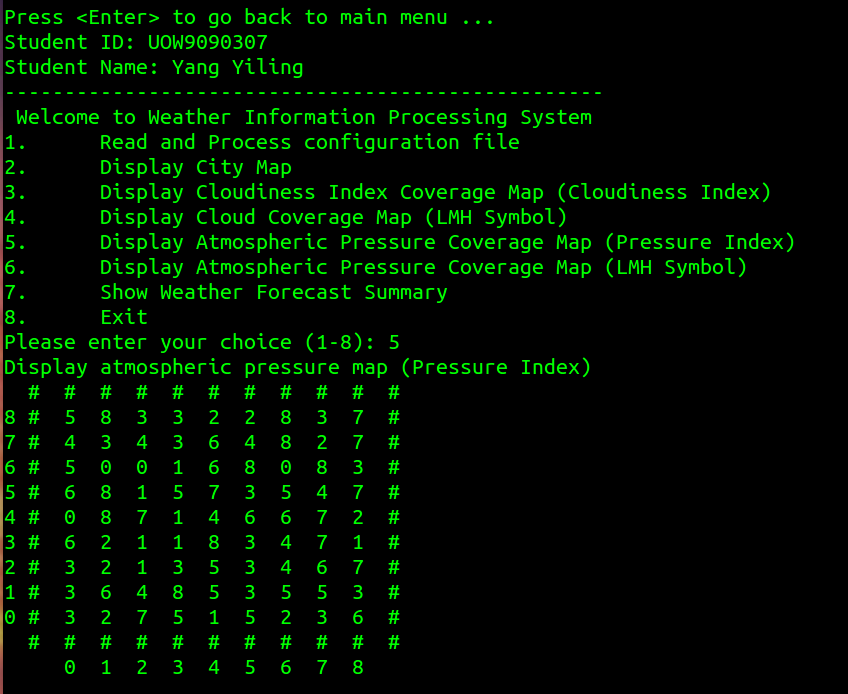


# Option 3 and 5:

Option 3 and 5 will display the cloudiness index /pressure index. The code loops through each cell in the grid, processing them from bottom to top for the y-axis. It mainly relies on the indMapPrintCell function to format the grid. This index is derived from cloud coverage value or atmospheric pressure value, depending on the isCloud parameter. For cloud coverage, the index is calculated as (cloudCover - 1) / 10, and the result is formatted with padding to ensure proper alignment in the grid.

Output on Ubuntu:

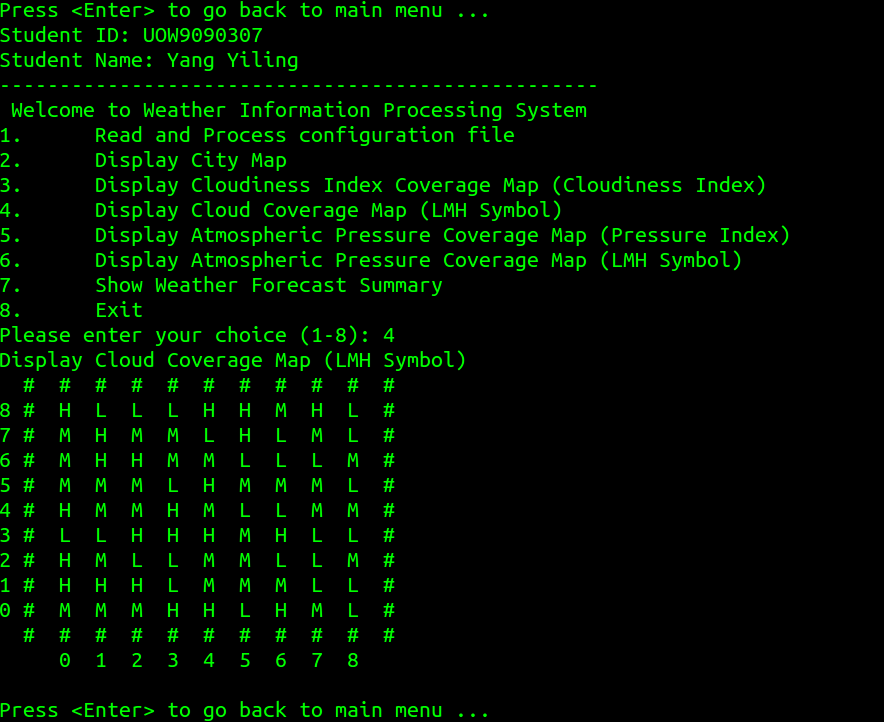


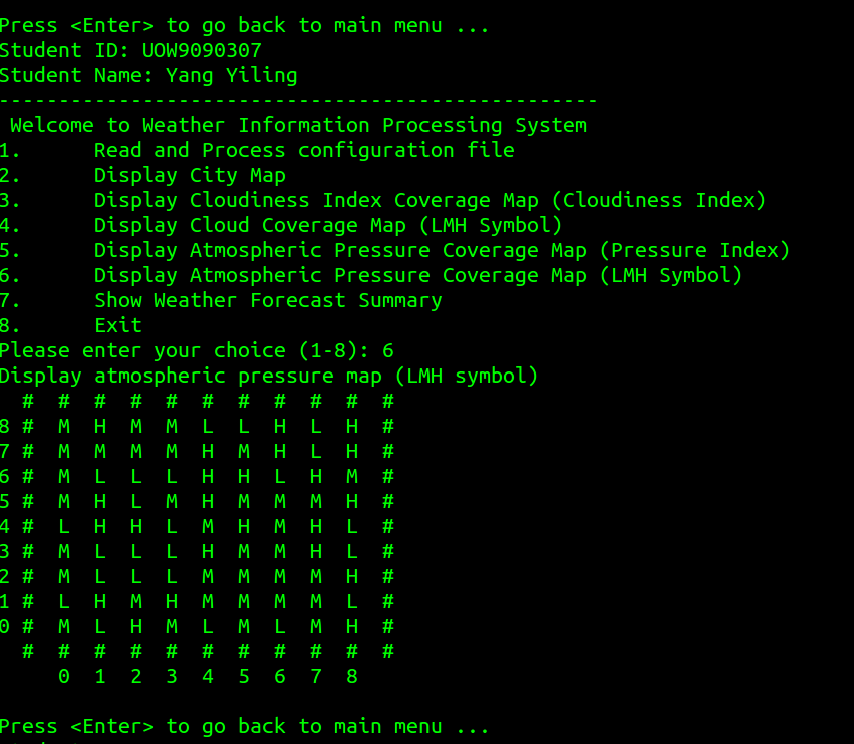


# Option 4 and 6:

Option 4 and 6 will show the LMH symbol. The LMH symbol is grouped by L for Low (cloud coverage < 35), M for Medium (cloud coverage >= 35 and < 65), H for High (cloud coverage >= 65), it is being stored in the lmhMapPrintCell string. By calling grid[x][y].lmhMapPrintCell(true), the code retrieves the LMH symbol for each cell. The convertToLMHSymbol method is used internally to classify the value into an LMH symbol. The resulting grid is displayed with borders and labels, providing a clear and intuitive visualization of cloud coverage levels across the entire area.

Output on Ubuntu:



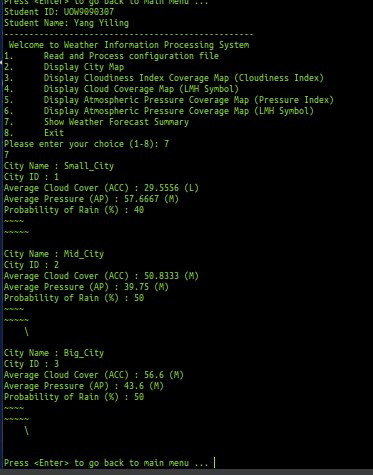


# Option 7:

Option 7 in the application is Show Weather Forecast Summary, which provides description of each city's weather conditions. This option calculates and displays crucial parameters such as average cloud cover (ACC), average atmospheric pressure (AP), and probability of rain (%). The function obtains the name and ID of each city, calculates the average cloud cover and atmospheric pressure, and uses predetermined thresholds to convert these results into LMH symbols (Low, Medium, High), and determines rain probability based on these symbols. For example, Low pressure (L) and High cloud cover (H) result in a 90% chance of rain. The summary is presented clearly, showing city name, ID, ACC, AP, and rain probability, making it easy to understand weather conditions across cities.

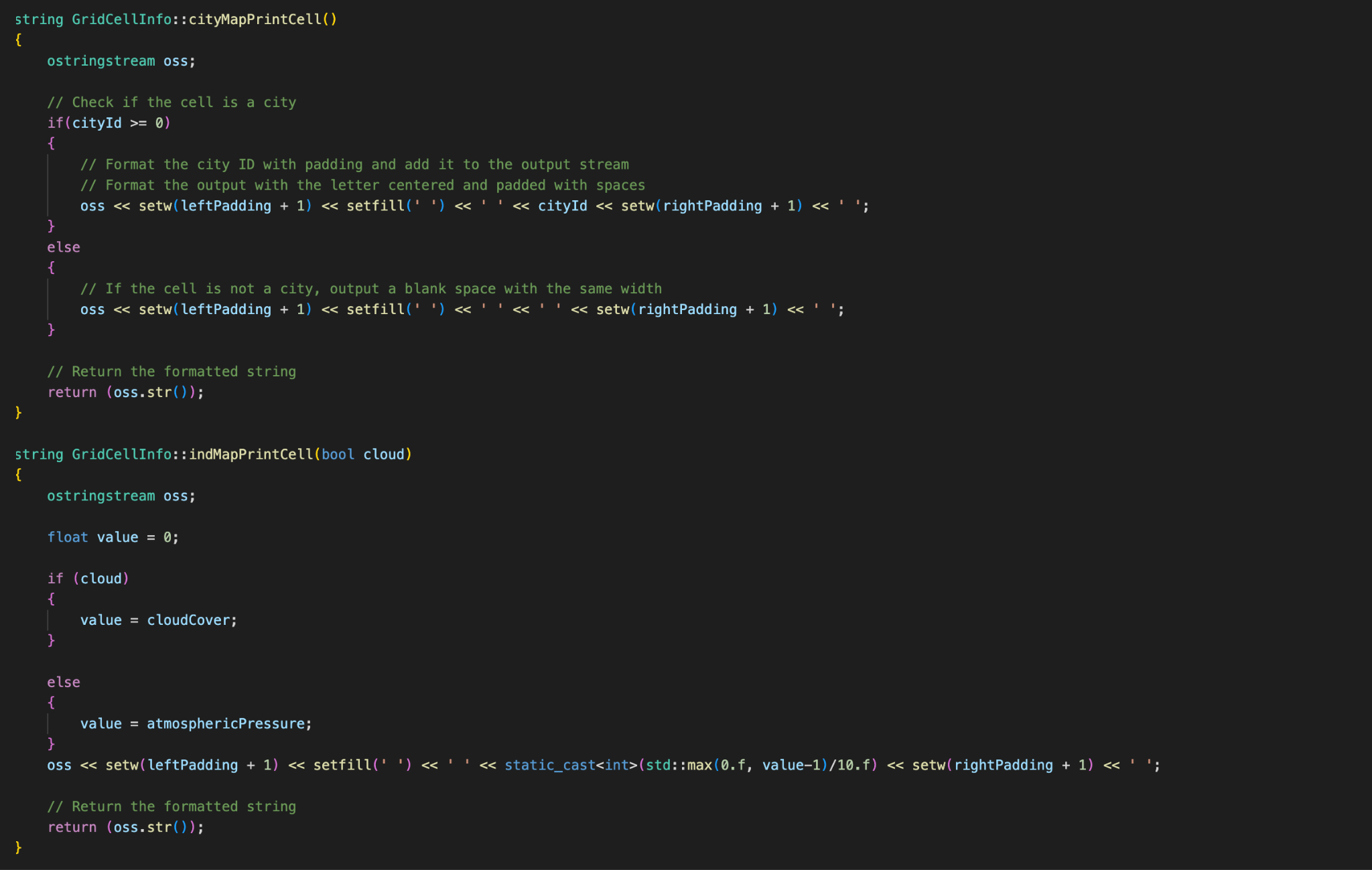


Output on Ubuntu:

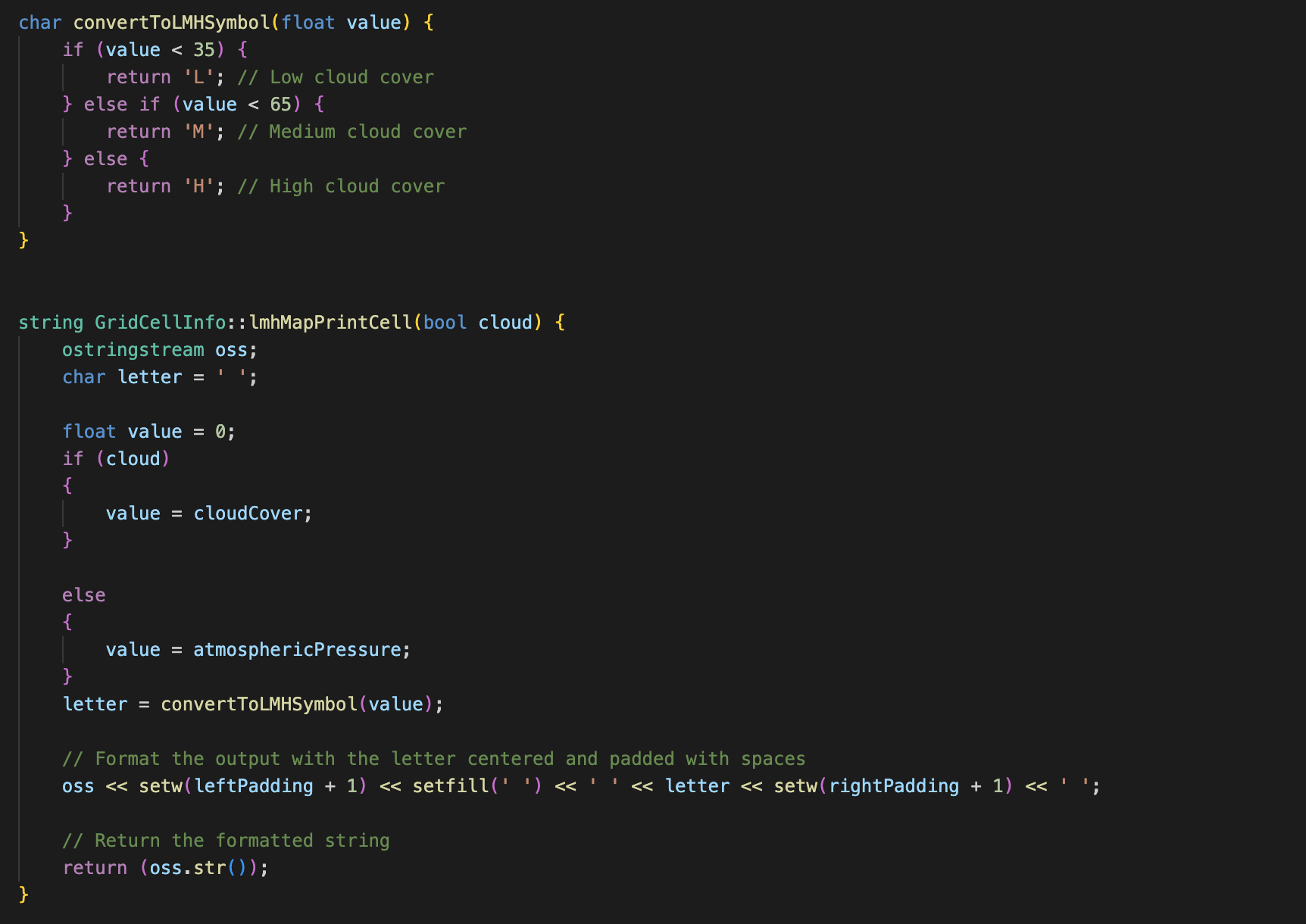


# 

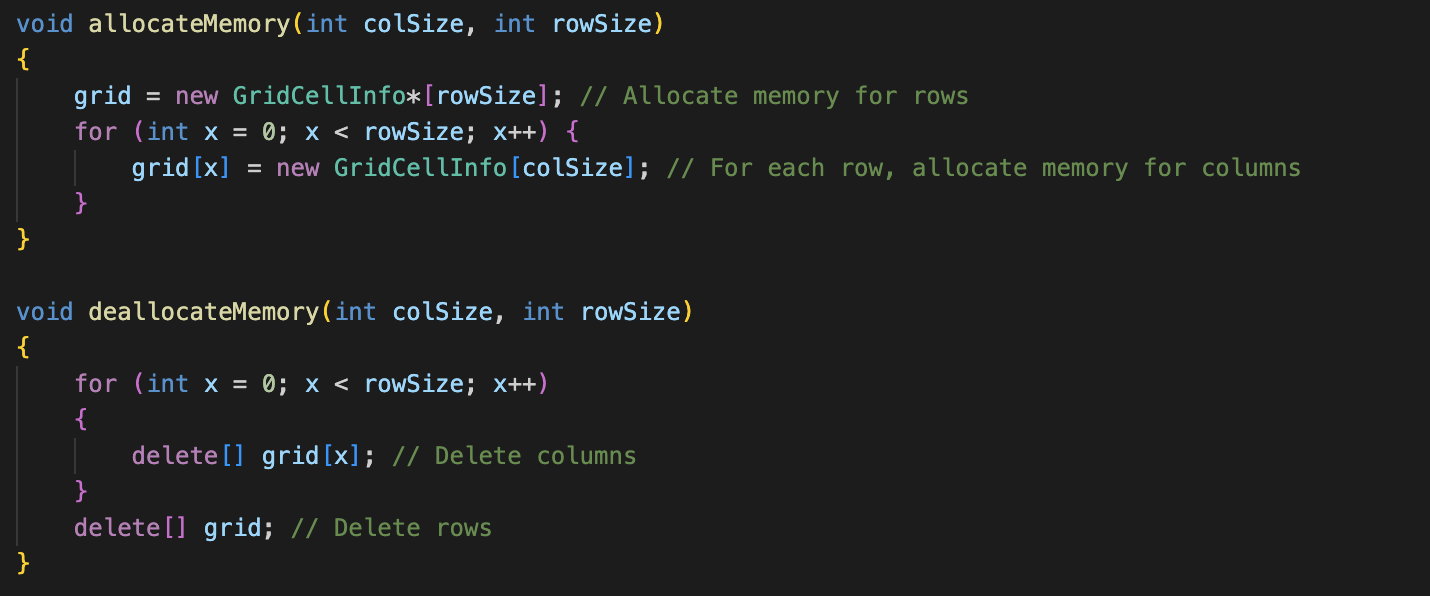
# Functions Used



The GridCellInfo struct is a flexible and efficient data structure designed to manage and display information about individual cells in a grid-based system, such as a map representing cities and non-city areas. It stores essential data like whether a cell is a city, the city's ID, atmospheric pressure, and cloud cover, allowing for detailed analysis and visualization. Struct also includes static members, such as numberOfDigits, leftPadding, and rightPadding, to ensure consistent formatting when printing the grid in various formats, including city maps, LMH (Low, Medium, High) maps, and individual value maps. By providing member functions like cityMapPrintCell, lmhMapPrintCell, and indMapPrintCell, Struct enables easy and customizable display of cell information.



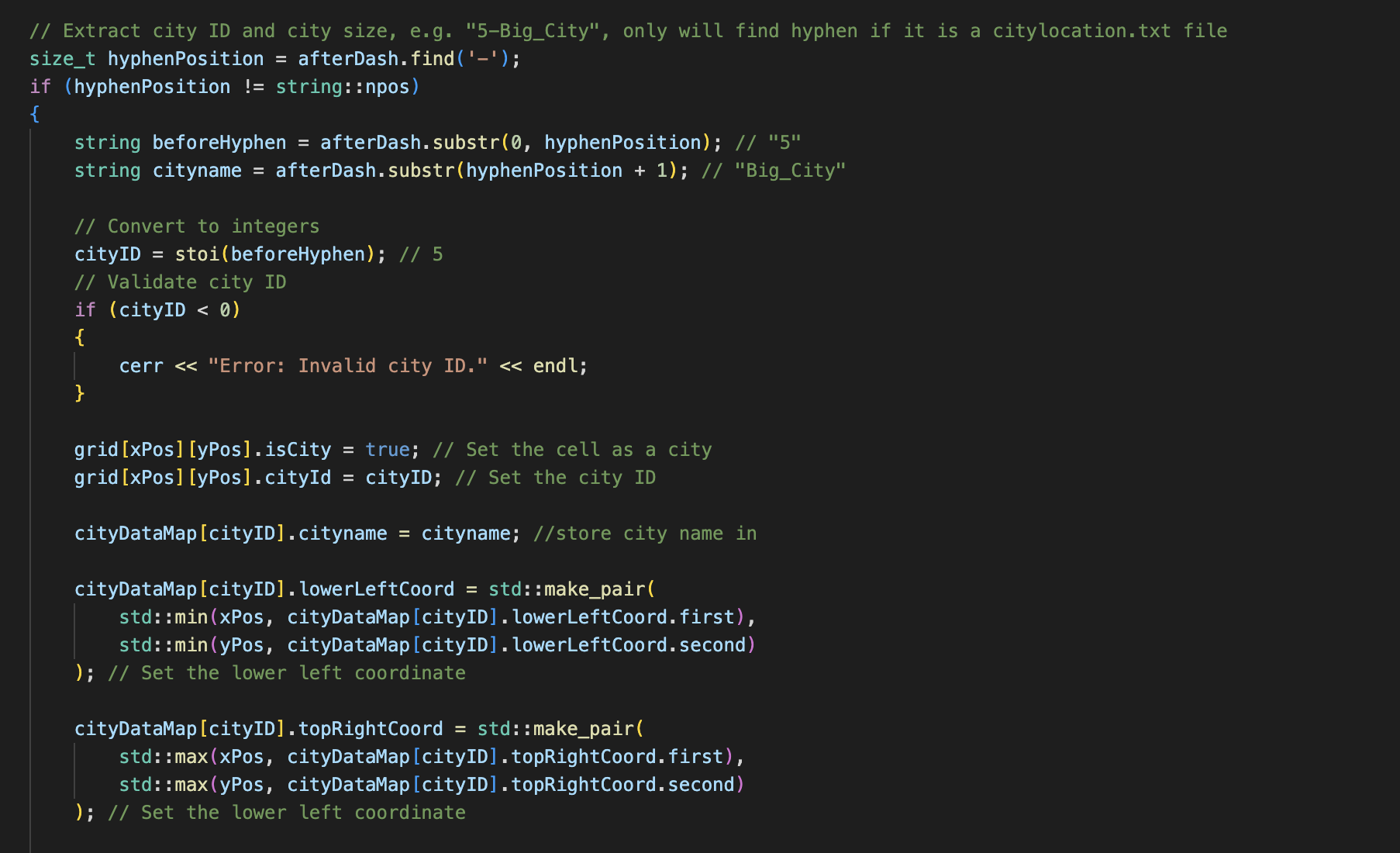
The provided code consists of two functions: convertToLMHSymbol and GridCellInfo::lmhMapPrintCell. The convertToLMHSymbol function takes a floating-point value as input and categorizes it into three levels—Low (L), Medium (M), or High (H)—based on predefined thresholds. Where value less than 35 is classified as L, between 35 and 65 as M, and 65 or above as H. This function is used to convert numerical values into symbolic representations for easier interpretation. The GridCellInfo::lmhMapPrintCell function, on the other hand, generates a formatted string for displaying the LMH symbol in a grid cell. It determines whether to use cloudCover or atmosphericPressure based on the cloud parameter, converts the selected value into an LMH symbol using convertToLMHSymbol, and then formats the output with appropriate padding on the left and right to ensure proper alignment. The formatted string is returned as the final output, making it suitable for displaying in a grid-based map.



The `allocateMemory` function dynamically allocates memory for a 2D grid of `GridCellInfo` objects by first creating an array of pointers (`grid`) to represent the rows, with each pointer pointing to an array of `GridCellInfo` objects for the columns. It iterates through each row and allocates memory for the corresponding columns, ensuring the grid is properly initialized with the specified dimensions (`rowSize` and `colSize`). The `deallocateMemory` function safely frees the allocated memory by first deleting each row's column array and then deleting the row array itself, preventing memory leaks. Together, these functions manage the 2D grid, ensuring efficient memory usage and cleanup.



The ProcessCityData is used to extract the coordinates from the text file, and updates the grid and city data accordingly. It first extracts and validates the coordinates (xPos, yPos) from the input line, adjusting them to fit within the grid's bounds. If the file is citylocation.txt, it further extracts the city ID and size, marks the grid cell as a city, and updates the city's boundaries in cityDataMap. For cloudcover.txt or pressure.txt, it extracts and validates the grid value (cloud cover or atmospheric pressure) and assigns it to the corresponding grid cell. The function ensures data integrity by performing checks for invalid values or out-of-bounds coordinates and provides error messages when issues are detected.

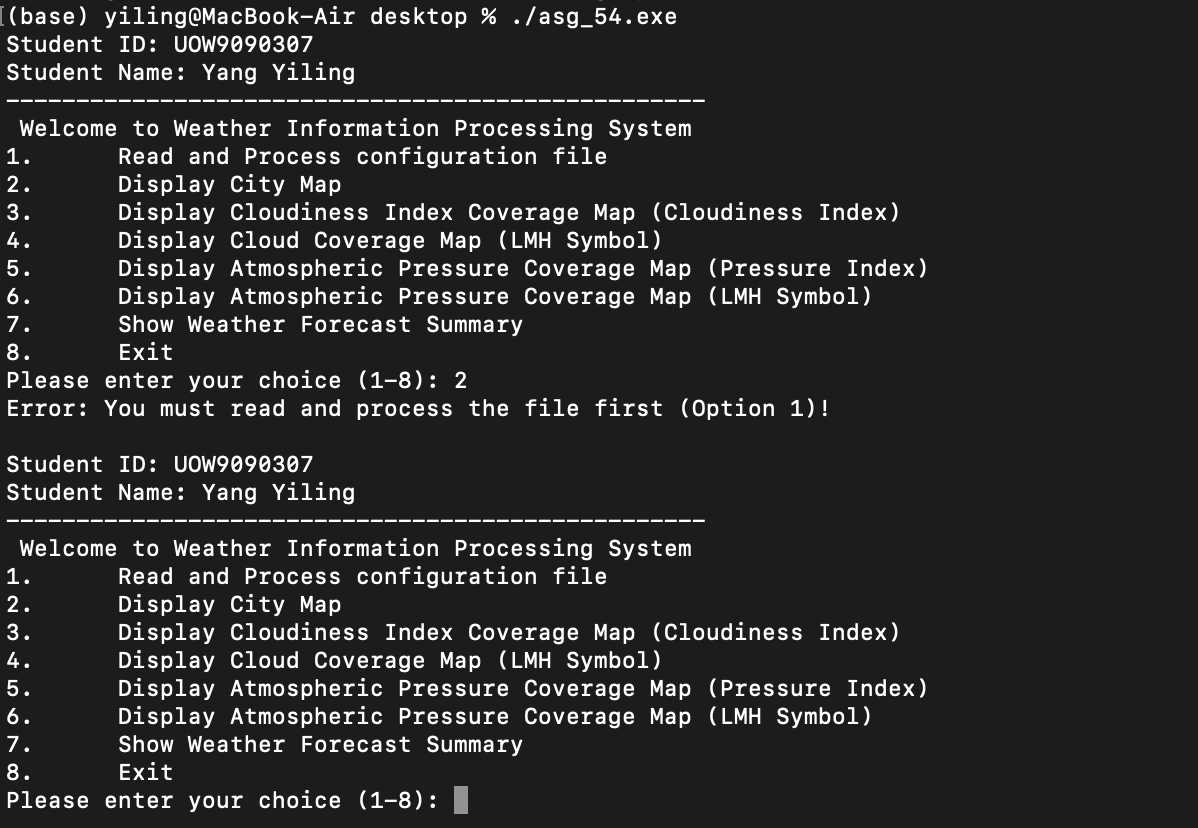


This code is used to store the cityname and cityid to be printed out in the display city summary for option 7.

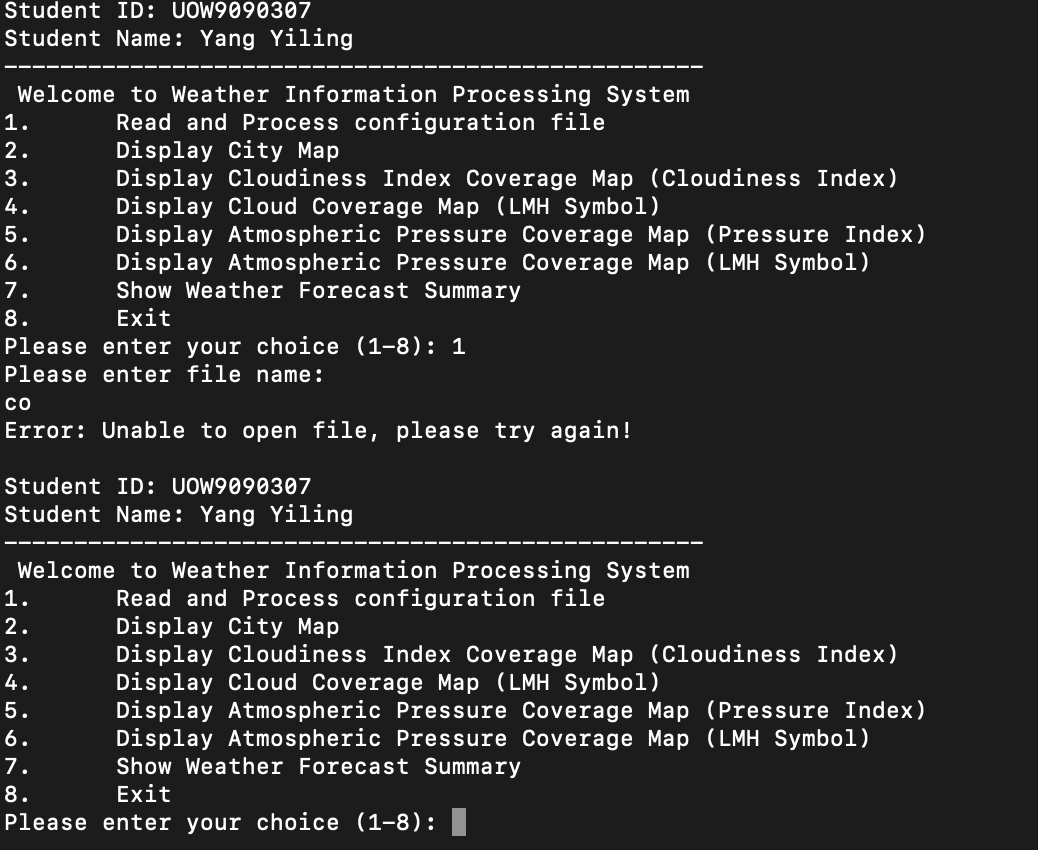


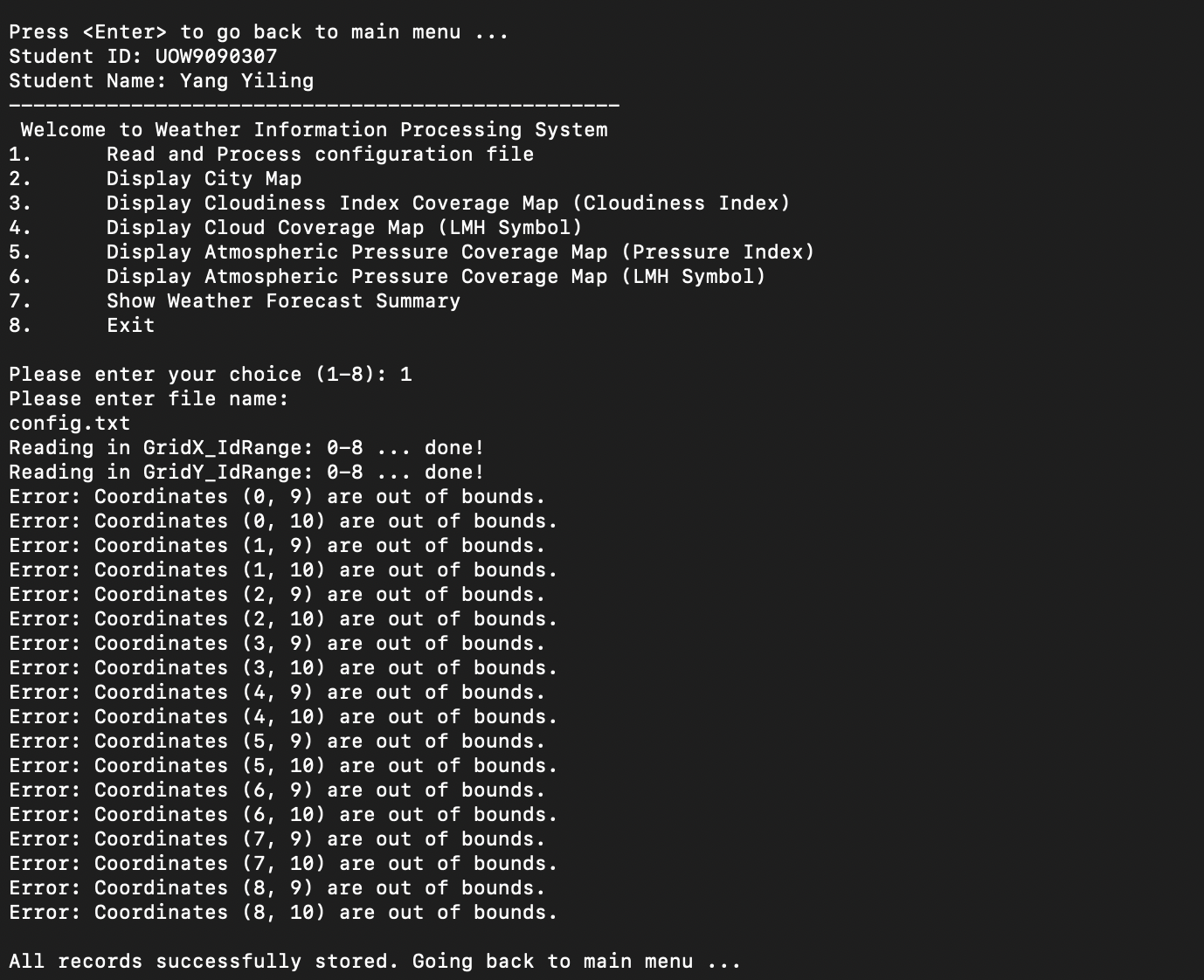
The printArray function displays a 2D array (array\_data) in a grid format, framed by borders and labeled axes, based on the specified option. It calculates the grid's dimensions using gridXmin, gridXmax, gridYmin, and gridYmax, then prints a top border. The function iterates through each cell of the array, printing its value or a transformed version depending on the option. For example, it prints raw values for option "1", divides values by 10 for options "2" and "4", or converts values to LMH symbols for options "3" and "5". Empty cells (value 0) are displayed as spaces. After printing the grid, it adds a bottom border and labels the x-axis with column indices for clarity. This ensures the array is presented in a structured and readable format, tailored to the chosen display option.

# Additional Features:



One function implemented is prompting the user to select option 1 first else there will not be any files that are passed through for the program to store.This is a validation which is required in order to run the program smoothly.

One feature implemented when the user enters the wrong file name, it will prompt the user that the file entered is wrong and will require them to try again from the main menu.



When reading coordinates it will alert users that coordinates are out of bond as it will abstract the reading of x-axis range and y-axis range when reading the config file. Thus if the value stored in the cloud/pressure/city location file exceeds the range, it will alert users of the coordinates that are out of bound.