**Advance Programming**

**CSCI 251**

**Assignment 1**

**Name: Jeslyn Ho Ka Yan**

**SIM ID/ UOW ID: 10241485/ 8535383**

**Tutorial Group Number: Group 6**

**Table of Content**

1 Assigment Overview 3

2 Key Components of the Program: 3

2.1 Data Structures and Prototypes: 3

2.2 Main Features of the Program: 4

3 Weather Forecast Summary Report: 6

# Assigment Overview

The assignment focuses on developing a Weather Information Processing System using C++. The primary objective is to read, process, and display weather-related data, including city locations, cloud coverage, and atmospheric pressure. The program also provides a weather forecast summary report based on the processed data.

# Key Components of the Program:

## Data Structures and Prototypes:

1. **A City structure is defined to store city-related information:**

struct City {

int x; // X-coordinate of the city

int y; // Y-coordinate of the city

int city\_id; // Unique identifier for the city

string city\_name; // Name of the city

};

This structure allows efficient storage and retrieval of city data. Each city’s information is uniquely identified by its city\_id, and its position on a grid is specified by the x and y coordinates.

1. **Function prototypes ensure modularity and readability. Notable prototypes include:**

bool readCityLocations(const string& filename, vector<City>& cities);

void displayCityMap(int gridXRange, int gridYRange, const vector<City>& cities);

bool readCloudCoverage(const string& filename, vector<vector<int>>& cloudCoverage, int gridXRange, int gridYRange);

pair<double, char> calculateACC(const vector<vector<int>>& cloudCoverage, const vector<City>& cities, int cityID, int gridXRange, int gridYRange);

## Main Features of the Program:

1. **Display Main Menu**

The displayMainMenu function presents a user-friendly interface with numbered options. This ensures users can navigate through different features conveniently.

void displayMainMenu() {

cout << "\nStudent ID : 8535383\n";

cout << "Student Name : Jeslyn Ho Ka Yan\n";

cout << "-----------------------------------------------\n";

cout << "Welcome to Weather Information Processing System!\n";

cout << "\n1) Read in and process a configuration file\n";

cout << "2) Display city map\n";

cout << "3) Display cloud coverage map (cloudiness index)\n";

cout << "4) Display cloud coverage map (LMH symbols)\n";

cout << "5) Display atmospheric pressure map (pressure index)\n";

cout << "6) Display atmospheric pressure map (LMH symbols)\n";

cout << "7) Show weather forecast summary report\n";

cout << "8) Quit\n\n";

}

The function is called in a loop within the main function, ensuring continuous interaction until the user chooses to quit.

1. **Reading Configuration File**

The readConfigFile function is responsible for initializing grid ranges and file paths. Here’s a step-by-step explanation:

* **File Opening:** It first attempts to open the specified configuration file. If unsuccessful, an error message is displayed.
* **Reading Grid Ranges:** It looks for lines containing GridX\_IdxRange and GridY\_IdxRange. These lines are parsed to extract the upper limits of the grid.
* **Identifying File Paths:** It identifies and stores the filenames for city locations, cloud coverage, and pressure data. Example:

if (line.find("GridX\_IdxRange") != string::npos) {

string range = line.substr(line.find("=") + 1);

size\_t dashPos = range.find("-");

int upperBound = stoi(range.substr(dashPos + 1));

gridXRange = upperBound;

}

1. **Reading City Locations**

The readCityLocations function processes city data from a file. Each line in the file follows the format [x,y]-id-name.

* **Splitting Strings:** The line is split into coordinate, ID, and name components using the splitString utility.
* **Validating and Storing Data:** After extracting the components, the function validates the coordinates and stores them in a vector of City structures. Example:

vector<string> parts = splitString(line, "-");

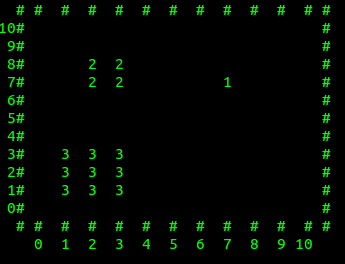
int x = stoi(parts[0]);

int y = stoi(parts[1]);

cities.push\_back({x, y, city\_id, city\_name});

1. **Displaying City Map**

The displayCityMap function generates a visual grid showing city locations. A 2D vector is initialized and populated based on city coordinates. Example Output:



1. **Reading Cloud Coverage and Pressure Data**

The readCloudCoverage function processes cloud coverage or pressure data in the format [x,y]-value.

* It reads each line, extracts coordinates and values, and validates them before storing them in a 2D vector. Example:

vector<string> parts = splitString(line, "-");

int x = stoi(parts[0]);

int y = stoi(parts[1]);

cloudCoverage[y][x] = stoi(parts[2]);

1. **Displaying Cloud Coverage and Pressure Maps**

The functions displayCloudCoverageIndex and displayPressureIndex print numerical indices for cloud coverage and pressure, respectively. Similarly, displayCloudCoverageLMH and displayPressureLMH display maps using symbols (L, M, H).

# Weather Forecast Summary Report:

1. **Calculating Average Cloud Cover and Pressure** The calculateACC and calculateAP functions compute averages for cloud cover and pressure, respectively. A bounding box is defined around the city, and values within the box are averaged. Example:

double sum = accumulate(values.begin(), values.end(), 0);

double avg = sum / values.size();

char symbol = (avg < 35) ? 'L' : (avg < 65) ? 'M' : 'H';

1. **Determining Probability of Rain** The determineRainProbability function uses predefined rules to calculate the probability of rain based on the ACC and AP symbols. Example Rule:

if (apSymbol == 'L' && accSymbol == 'H') {

probability = 90;

}

1. **Generating the Report**

The displayWeatherForecastSummary function iterates through all cities, calculates weather metrics, and displays a formatted summary. Example Output:

City Name : CityA

City ID : 1

Ave. Cloud Cover (ACC) : 42.50 (M)

Ave. Pressure (AP) : 60.75 (M)

Probability of Rain (%) : 50.00

~~~~

~~~~~

\

**Conclusion:**

This program effectively demonstrates advanced programming concepts such as file handling, data processing, and modular function design. The Weather Information Processing System provides a comprehensive solution for visualizing and analyzing weather data, meeting the assignment's requirements comprehensively.