TASK 01

Implementation Plan

Basic Level

Manage Locations

o Create a Location class with attributes for name, latitude, and longitude.

o Implement methods to add, remove, and list locations.

Define Weather Variables

o Create a WeatherVariable class to manage different weather parameters such as temperature, wind speed, etc.

o Implement methods to define and manage these variables.

Fetch and Display Weather Forecast Data

o Use an API (e.g., Open Meteo) to fetch weather forecast data.

o Implement a WeatherForecastingSystem class to handle API interactions and data retrieval.

o Display the weather forecast data to the user.

Implementation Plan

Historical Weather Data

o Implement a HistoricalWeatherSystem class to handle fetching historical weather data from an API.

o Extend the functionality to display historical data.

Export Results

o Implement methods to export weather forecast and historical data

to CSV and JSON formats

Air Quality Data

o Implement an AirQualityForecastingSystem class to fetch air quality data.

o Add methods to handle air quality data retrieval and display.

Advanced Level

Expert Level

Offline Mode

o Implement functionality to save search results to a cloud database (e.g., Google FireStore).

o Add methods to retrieve and display saved data, with filtering options by data category and parameters.

o Ensure the application can function in offline mode, using locally stored data when necessary.

Class Definitions

o Define classes for Location, WeatherVariable, WeatherForecastingSystem, HistoricalWeatherSystem, and AirQualityForecastingSystem.

o Use inheritance and polymorphism where appropriate to manage common functionalities.

API Integration

o Research and choose an appropriate weather and air quality data API.

o Implement methods in WeatherForecastingSystem, HistoricalWeatherSystem, and

AirQualityForecastingSystem to interact with these APIs.

o Handle API responses and errors gracefully.

Steps for Implementation

User Interface

o Create a console-based user interface to interact with the application.

o Implement functions to manage user inputs, validate data, and handle exceptions.

Data Export

o Implement methods to export data to CSV and JSON formats.

o Ensure that exported data is properly formatted and can be easily imported into other applications.

Offline Mode and Cloud Integration

o Implement functionality to save data to a cloud database like Google FireStore.

o Implement methods to retrieve and display saved data, with filtering options.

o Ensure the application can work offline using locally stored data when no internet connection is available.

Testing and Validation

o Test each functionality thoroughly to ensure it works as expected.

o Implement unit tests and integration tests to validate the application's behavior.

o Handle edge cases and ensure robust error handling.

Documentation

o Document the code and provide a user manual for the application.

o Include instructions on how to set up and run the application, as well as details on the functionalities available at each level.

Code Overview

This C++ program is a simple weather application that allows users to manage locations and fetch mock weather, historical weather, and air quality data for those locations. It also provides functionality to export location data. The program uses file I/O to save and load location data and employs standard C++ libraries for handling vectors, strings, and file systems.

Key Components

Libraries Used

- <**iostream**>: Provides input and output stream objects (cin, cout, etc.) for console I/O.
- <**vector**>: Offers dynamic array capabilities through the vector container, allowing storage and manipulation of a list of Location objects.
- <**string**>: Provides the string class to handle text data easily.
- <**fstream**>: Enables file I/O operations with ifstream and ofstream for reading and writing files, respectively.
- <**sstream**>: Facilitates string stream operations, useful for parsing strings and extracting data.
- <**algorithm**>: Supplies functions like remove_if for performing operations on collections.
- <**filesystem**>: Provides utilities for file system operations, such as creating directories.
- <**cstdlib**>: Includes general-purpose functions like rand() for generating random numbers.
- <**ctime**>: Supplies functions related to date and time, used here for seeding the random number generator with the current time.

Classes

1. **Location**:
    - Represents a geographical location with a name, latitude, and longitude.
    - **Attributes**:
        - name: A string representing the location's name.
        - latitude: A double indicating the latitude of the location.
        - longitude: A double indicating the longitude of the location.
    - **Methods**:
        - getName(): Returns the name of the location.
        - getLatitude(): Returns the latitude of the location.
        - getLongitude(): Returns the longitude of the location.
2. **Location Manager**:
    - Manages a collection of Location objects, with the ability to add, remove, list, save, and load locations from a file.
    - **Attributes**:
        - locations: A vector storing Location objects.
        - filePath: A string specifying the path to the file where location data is stored.
    - **Methods**:

- createDirectoryIfNotExists(const string& dirPath): Ensures that the specified directory exists, creating it if necessary.
- addLocation(const string& name, double latitude, double longitude): Adds a new location to the list and saves the list to a file.
- removeLocation(const string& name): Removes a location by name and updates the file.
- listLocations() const: Displays all stored locations.
- getLocations() const: Returns the current list of locations.
- saveToFile() const: Writes the current list of locations to a file.
- loadFromFile(): Loads locations from a file into the locations vector.

3. **DataExporter:**
   - Provides functionality to export location data.
   - **Methods**:
     - exportData(const vector<Location>& locations): Exports location data to an external format (currently simulated with console output).

Functions

- **generateRandomWeatherData()**:
  - Generates random weather data, including temperature and weather conditions, simulating a real weather API response.
- **generateRandomHistoricalData()**:
  - Generates random historical weather data, simulating fetching historical weather information for a specific date range.
- **generateRandomAirQualityData()**:
  - Generates random air quality index (AQI) data, simulating air quality information retrieval.
- **mockFetchWeatherData(double latitude, double longitude)**:
  - Simulates fetching current weather data based on latitude and longitude by calling generateRandomWeatherData().
- **mockFetchHistoricalData(double latitude, double longitude, const string& startDate, const string& endDate)**:
  - Simulates fetching historical weather data for a location and date range using generateRandomHistoricalData().
- **mockFetchAirQualityData(double latitude, double longitude)**:
  - Simulates fetching air quality data based on latitude and longitude using generateRandomAirQualityData().

Main Function

- **main()**:
  - The entry point of the application, providing a menu-driven interface for users to interact with the application.
  - **Operations**:
    - Presents a menu to the user with options to add, remove, list locations, fetch weather, historical data, air quality data, export data, or exit the application.

- **Case 1**: Adds a new location by prompting the user for location name, latitude, and longitude.
- **Case 2**: Removes an existing location by prompting the user for the location name.
- **Case 3**: Lists all stored locations.
- **Case 4**: Fetches and displays mock weather data for a given latitude and longitude.
- **Case 5**: Fetches and displays mock historical weather data for a given latitude, longitude, and date range.
- **Case 6**: Fetches and displays mock air quality data for a given latitude and longitude.
- **Case 7**: Exports location data using DataExporter.
- **Case 8**: Exits the application.

File Handling

- **File Path**: "C:\\Users\\USER\\myapp\\locations.txt"
  - The program saves and loads location data to and from a file located at the specified path.

Random Data Generation

- The program uses random data generation to simulate API responses for weather, historical weather, and air quality data. This is done to mimic real-world behavior without actual API integration.

CODE:

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <filesystem>
#include <cstdlib>
#include <ctime>

using namespace std;

class Location {
private:
    string name;
    double latitude;
```

```
    double longitude;

public:
    Location(const string& name, double latitude, double longitude)
        : name(name), latitude(latitude), longitude(longitude) {}

    string getName() const {
        return name;
    }

    double getLatitude() const {
        return latitude;
    }

    double getLongitude() const {
        return longitude;
    }
};

class LocationManager {
private:
    vector<Location> locations;
    string filePath = "C:\\Users\\USER\\myapp\\locations.txt"; // Specify your custom directory

    void createDirectoryIfNotExists(const string& dirPath) {
        filesystem::create_directories(dirPath);
    }

public:
    LocationManager() {
        // Seed the random number generator
        srand(static_cast<unsigned>(time(0)));

        // Ensure directory exists
        createDirectoryIfNotExists("C:\\Users\\USER\\myapp");
        loadFromFile();
    }

    void addLocation(const string& name, double latitude, double longitude) {
        locations.emplace_back(name, latitude, longitude);
        cout << "Location added: " << name << endl;
        saveToFile();
    }

    void removeLocation(const string& name) {
```

```
    auto it = remove_if(locations.begin(), locations.end(), [&](const Location& loc) {
        return loc.getName() == name;
    });

    if (it != locations.end()) {
        locations.erase(it, locations.end());
        cout << "Location removed: " << name << endl;
        saveToFile();
    } else {
        cout << "Location not found: " << name << endl;
    }
}

void listLocations() const {
    if (locations.empty()) {
        cout << "No locations available." << endl;
        return;
    }

    for (const auto& location : locations) {
        cout << "Location: " << location.getName()
             << ", Latitude: " << location.getLatitude()
             << ", Longitude: " << location.getLongitude() << endl;
    }
}

const vector<Location>& getLocations() const {
    return locations;
}

void saveToFile() const {
    ofstream file(filePath);
    if (file.is_open()) {
        for (const auto& loc : locations) {
            file << loc.getName() << "," << loc.getLatitude() << "," << loc.getLongitude() << "\n";
        }
        file.close();
    } else {
        cerr << "Unable to open file for writing: " << filePath << endl;
    }
}

void loadFromFile() {
    ifstream file(filePath);
    if (file.is_open()) {
```

```
        string line;
        while (getline(file, line)) {
            istringstream ss(line);
            string name;
            double latitude, longitude;
            getline(ss, name, ',');
            ss >> latitude;
            ss.ignore(1); // Ignore the comma
            ss >> longitude;
            locations.emplace_back(name, latitude, longitude);
        }
        file.close();
    } else {
        cerr << "Unable to open file for reading: " << filePath << endl;
    }
  }
};

// Function to generate a random temperature between 15°C and 35°C
string generateRandomWeatherData() {
  int temp = 15 + rand() % 21; // Random temperature between 15°C and 35°C
  string weather = (rand() % 2 == 0) ? "Clear sky" : "Partly cloudy";
  return "Temperature: " + to_string(temp) + "°C\nWeather: " + weather;
}

// Function to generate random historical weather data
string generateRandomHistoricalData() {
  int temp = 15 + rand() % 21; // Random temperature between 15°C and 35°C
  return "Date: 2024-08-01, Temp: " + to_string(temp) + "°C";
}

// Function to generate random air quality data
string generateRandomAirQualityData() {
  int aqi = 1 + rand() % 150; // Random AQI value between 1 and 150
  return "AQI: " + to_string(aqi);
}

// Mock function to simulate fetching weather data
string mockFetchWeatherData(double latitude, double longitude) {
  return generateRandomWeatherData();
}

// Mock function to simulate fetching historical weather data
string mockFetchHistoricalData(double latitude, double longitude, const string& startDate, const string&
endDate) {
```

```
    return generateRandomHistoricalData();
}

// Mock function to simulate fetching air quality data
string mockFetchAirQualityData(double latitude, double longitude) {
    return generateRandomAirQualityData();
}

class DataExporter {
public:
    void exportData(const vector<Location>& locations) {
        if (locations.empty()) {
            cout << "No data to export." << endl;
            return;
        }

        // Mock exporting data
        cout << "Exporting data..." << endl;
        for (const auto& location : locations) {
            cout << "Exported Location: " << location.getName() << endl;
        }
    }
};

int main() {
    LocationManager locationManager;
    DataExporter dataExporter;

    while (true) {
        cout << "Weather Application\n"
             << "1. Add Location\n"
             << "2. Remove Location\n"
             << "3. List Locations\n"
             << "4. Fetch Weather Data\n"
             << "5. Fetch Historical Data\n"
             << "6. Fetch Air Quality Data\n"
             << "7. Export Data\n"
             << "8. Exit\n"
             << "Enter your choice: ";

        int choice;
        cin >> choice;

        if (choice == 8) {
            cout << "Exiting application." << endl;
```

```
        break;
    }

    switch (choice) {
        case 1: {
            string name;
            double latitude, longitude;
            cout << "Enter location name: ";
            cin >> name;
            cout << "Enter latitude: ";
            cin >> latitude;
            cout << "Enter longitude: ";
            cin >> longitude;
            locationManager.addLocation(name, latitude, longitude);
            break;
        }
        case 2: {
            string name;
            cout << "Enter location name to remove: ";
            cin >> name;
            locationManager.removeLocation(name);
            break;
        }
        case 3: {
            locationManager.listLocations();
            break;
        }
        case 4: {
            double latitude, longitude;
            cout << "Enter latitude: ";
            cin >> latitude;
            cout << "Enter longitude: ";
            cin >> longitude;
            cout << mockFetchWeatherData(latitude, longitude) << endl;
            break;
        }
        case 5: {
            double latitude, longitude;
            string startDate, endDate;
            cout << "Enter latitude: ";
            cin >> latitude;
            cout << "Enter longitude: ";
            cin >> longitude;
            cout << "Enter start date (YYYY-MM-DD): ";
            cin >> startDate;
```

```
            cout << "Enter end date (YYYY-MM-DD): ";
            cin >> endDate;
            cout << mockFetchHistoricalData(latitude, longitude, startDate, endDate) << endl;
            break;
        }
        case 6: {
            double latitude, longitude;
            cout << "Enter latitude: ";
            cin >> latitude;
            cout << "Enter longitude: ";
            cin >> longitude;
            cout << mockFetchAirQualityData(latitude, longitude) << endl;
            break;
        }
        case 7: {
            // Export data using the public method getLocations
            dataExporter.exportData(locationManager.getLocations());
            break;
        }
        default:
            cout << "Invalid choice. Please try again." << endl;
}
    }

    return 0;
}
```

OUTPUT:

```
Weather Application
1. Add Location
2. Remove Location
3. List Locations
4. Fetch Weather Data
5. Fetch Historical Data
6. Fetch Air Quality Data
7. Export Data
8. Exit
Enter your choice: 1
```

```
Enter location name: delhi
Enter latitude: -111
Enter longitude: -180
Location added: delhi
```

```
Enter your choice: 3
Location: islamabad, Latitude: 45, Longitude: 150
Location: rawalpindi, Latitude: 34, Longitude: 120
Location: delhi, Latitude: -111, Longitude: -180
```

```
Enter your choice: 4
Enter latitude: -111
Enter longitude: -180
Temperature: 28°C
Weather: Partly cloudy
```

```
Enter your choice: 5
Enter latitude: -111
Enter longitude: -180
Enter start date (YYYY-MM-DD): 2022-07-09
Enter end date (YYYY-MM-DD): 2024-05-08
Date: 2024-08-01, Temp: 30°C
```

```
Enter your choice: 6
Enter latitude: 45
Enter longitude: 150
AQI: 19
```

```
Enter your choice: 7
Exporting data...
Exported Location: islamabad
Exported Location: rawalpindi
Exported Location: delhi
```

```
Enter your choice: 2
Enter location name to remove: delhi
Location removed: delhi
```

```
Enter your choice: 3
Location: islamabad, Latitude: 45, Longitude: 150
Location: rawalpindi, Latitude: 34, Longitude: 120
```

```
Enter your choice: 8
Exiting application.
```