Digital Empowerment Pakistan

Internship Batch 3 DEN

Domain:
C++ programming

task :01

Submitted By:

Laiba Husna

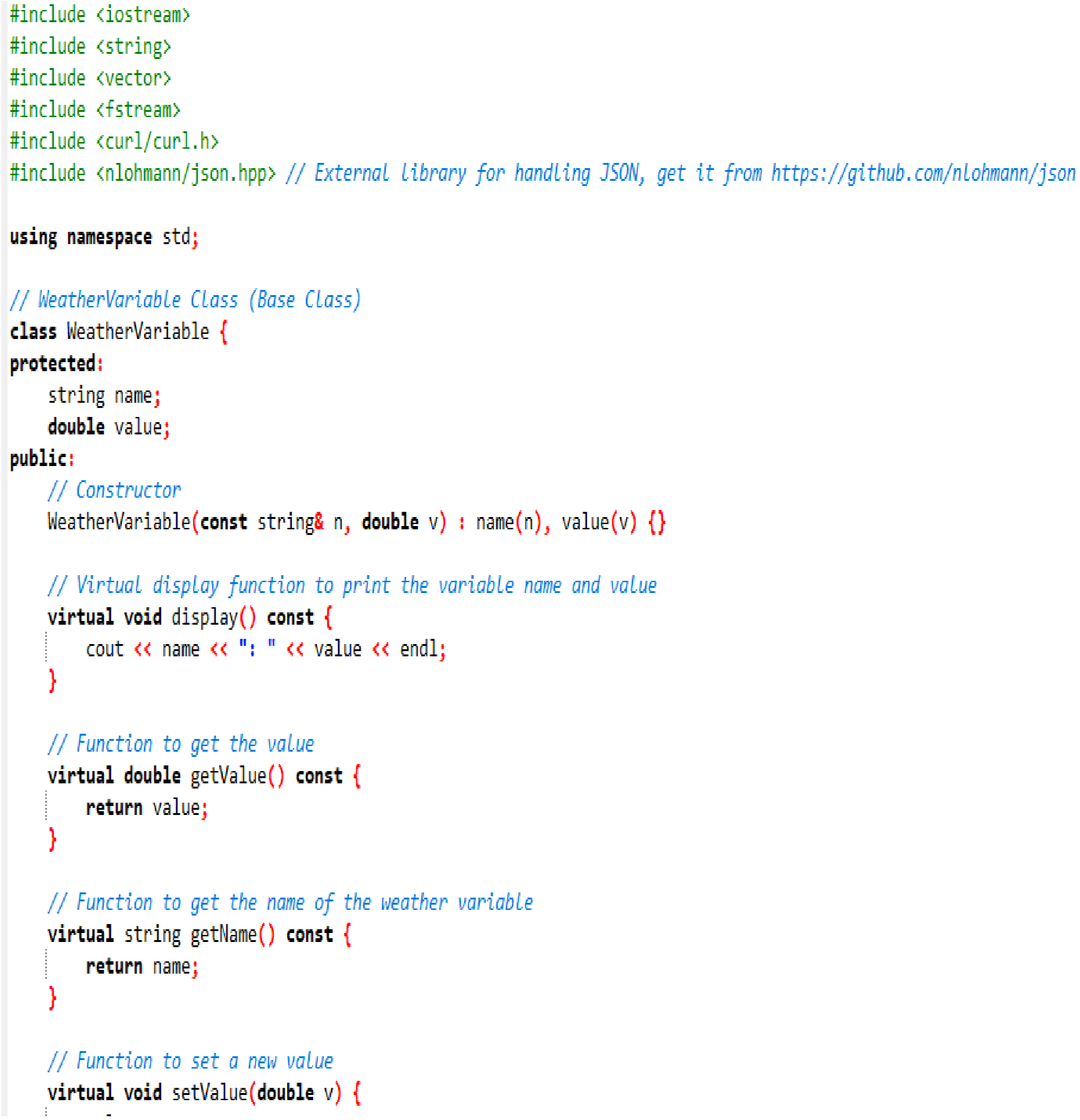
Task:

Define Weather Variables

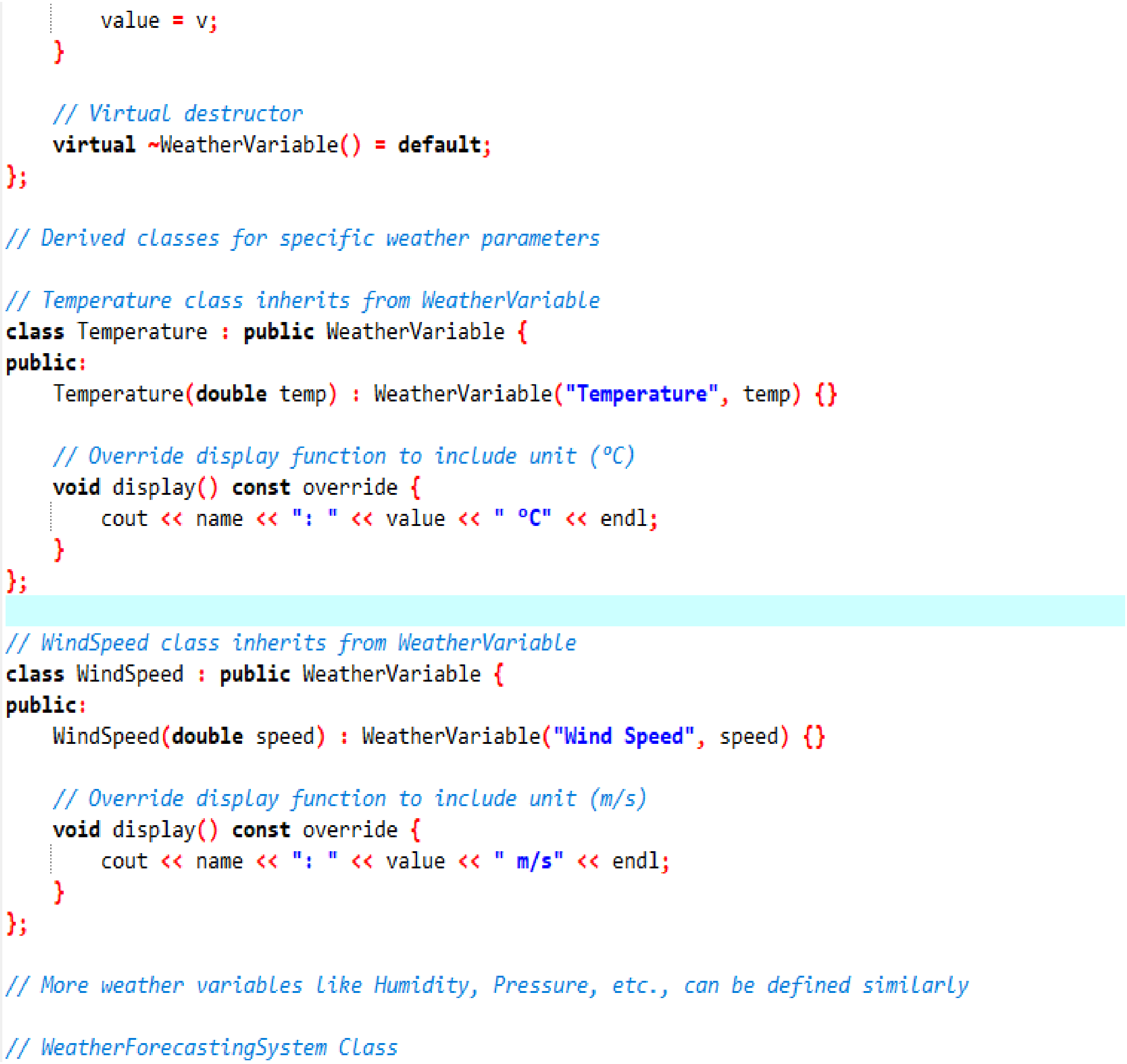
- **Create a WeatherVariable class to manage different weather parameters such as temperature, wind speed, etc.**
- **Implement methods to define and manage these variables.**

Solution:

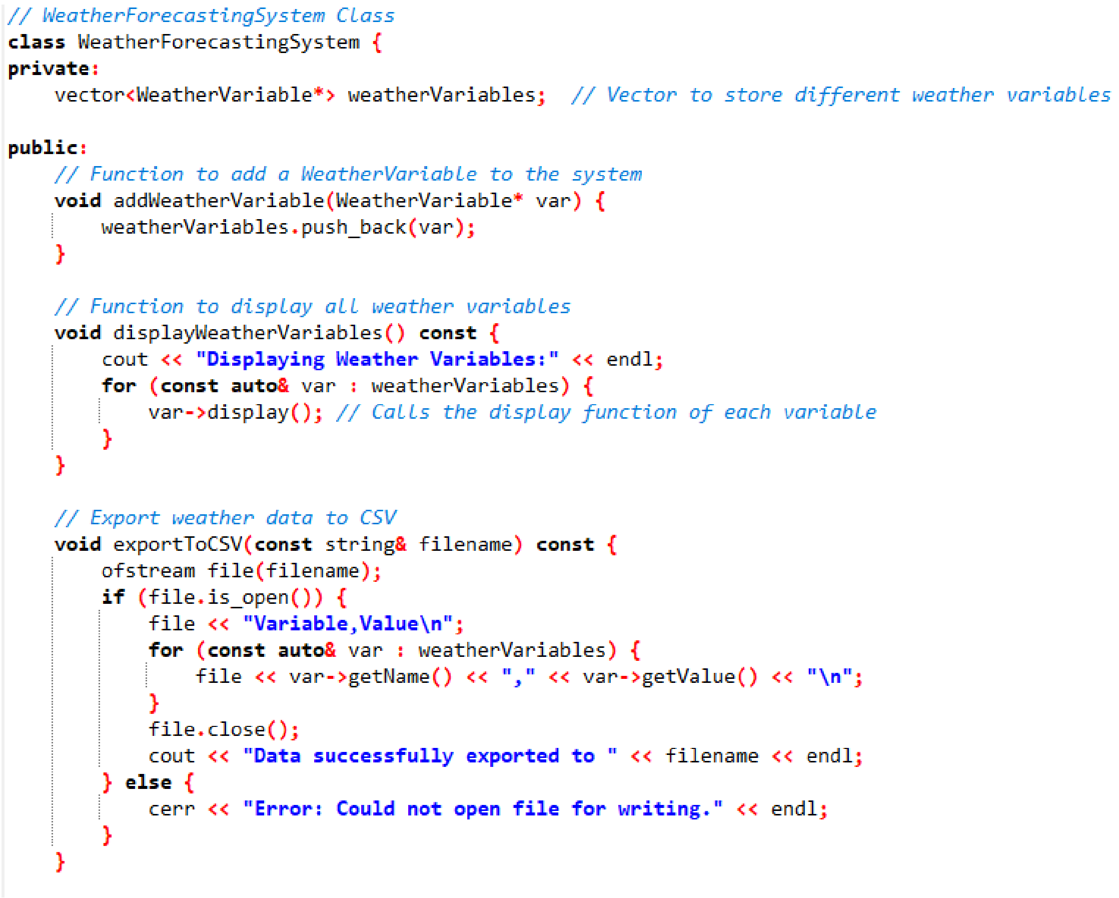
Code:



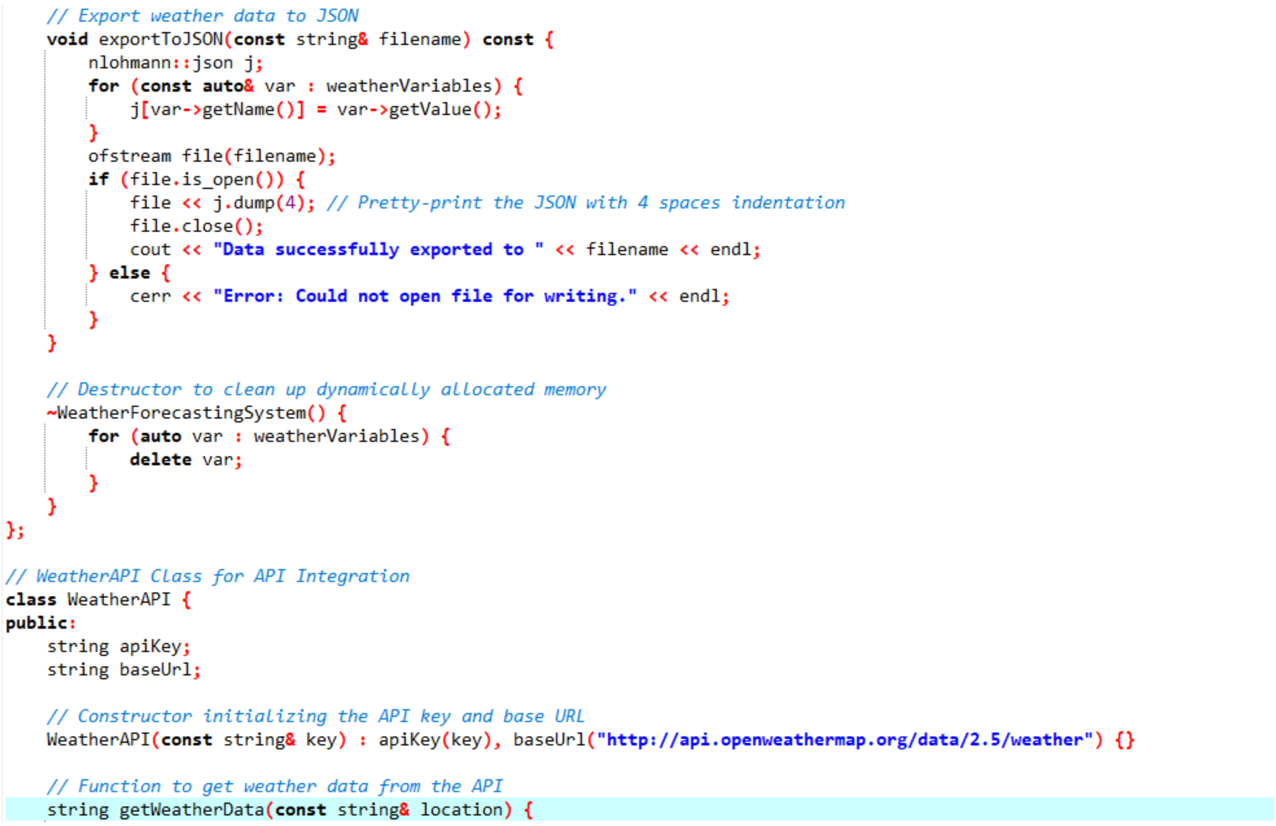
```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <curl/curl.h>
#include <nlohmann/json.hpp> // External library for handling JSON, get it from https://github.com/nlohmann/json

using namespace std;

// WeatherVariable Class (Base Class)
class WeatherVariable {
protected:
    string name;
    double value;
public:
    // Constructor
    WeatherVariable(const string& n, double v) : name(n), value(v) {}

    // Virtual display function to print the variable name and value
    virtual void display() const {
        cout << name << ": " << value << endl;
    }

    // Function to get the value
    virtual double getValue() const {
        return value;
    }

    // Function to get the name of the weather variable
    virtual string getName() const {
        return name;
    }

    // Function to set a new value
    virtual void setValue(double v) {
```



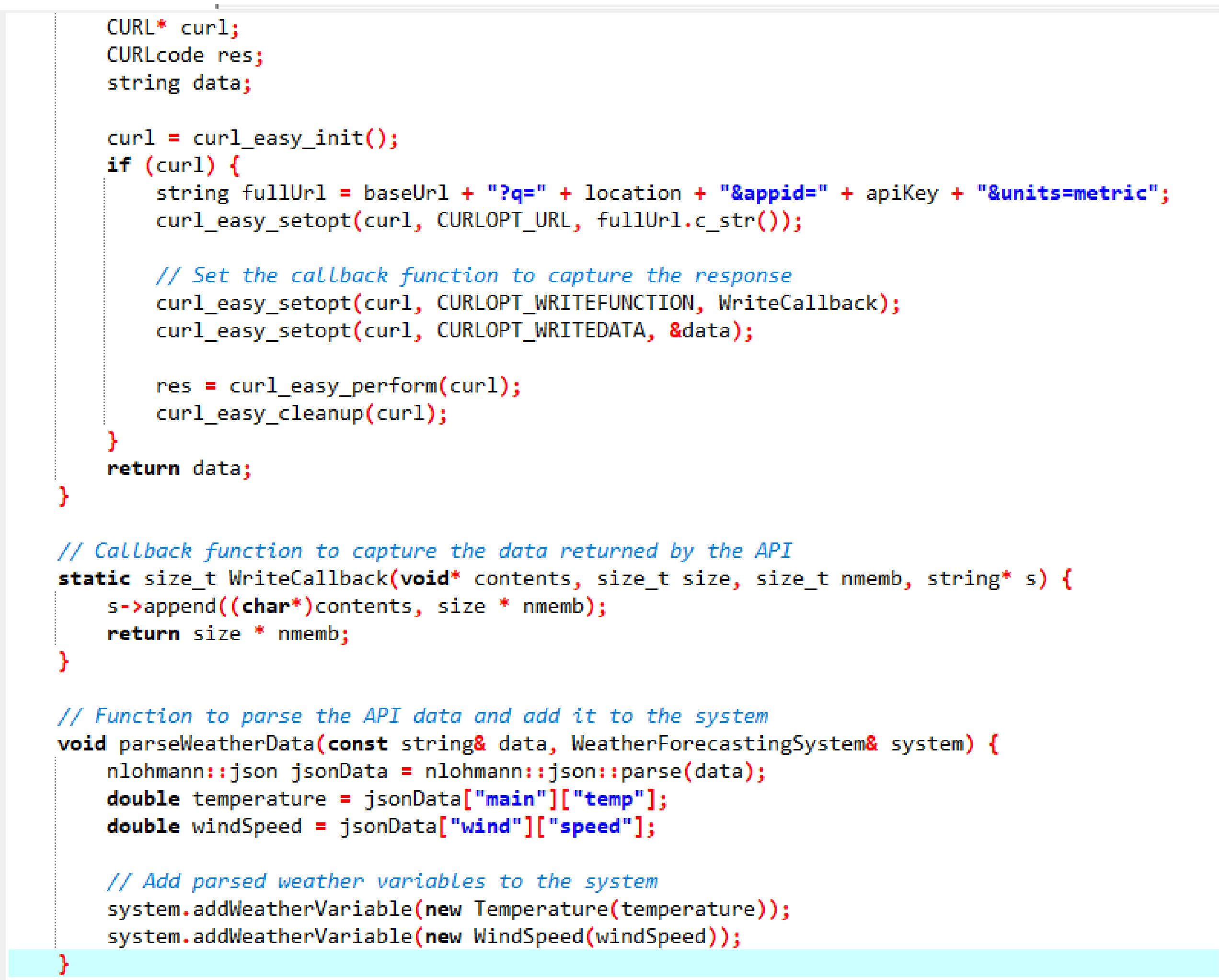
```
        value = v;
    }

    // Virtual destructor
    virtual ~WeatherVariable() = default;
};

// Derived classes for specific weather parameters

// Temperature class inherits from WeatherVariable
class Temperature : public WeatherVariable {
public:
    Temperature(double temp) : WeatherVariable("Temperature", temp) {}

    // Override display function to include unit (°C)
    void display() const override {
        cout << name << ": " << value << " °C" << endl;
    }
};

// WindSpeed class inherits from WeatherVariable
class WindSpeed : public WeatherVariable {
public:
    WindSpeed(double speed) : WeatherVariable("Wind Speed", speed) {}

    // Override display function to include unit (m/s)
    void display() const override {
        cout << name << ": " << value << " m/s" << endl;
    }
};

// More weather variables like Humidity, Pressure, etc., can be defined similarly

// WeatherForecastingSystem Class
```



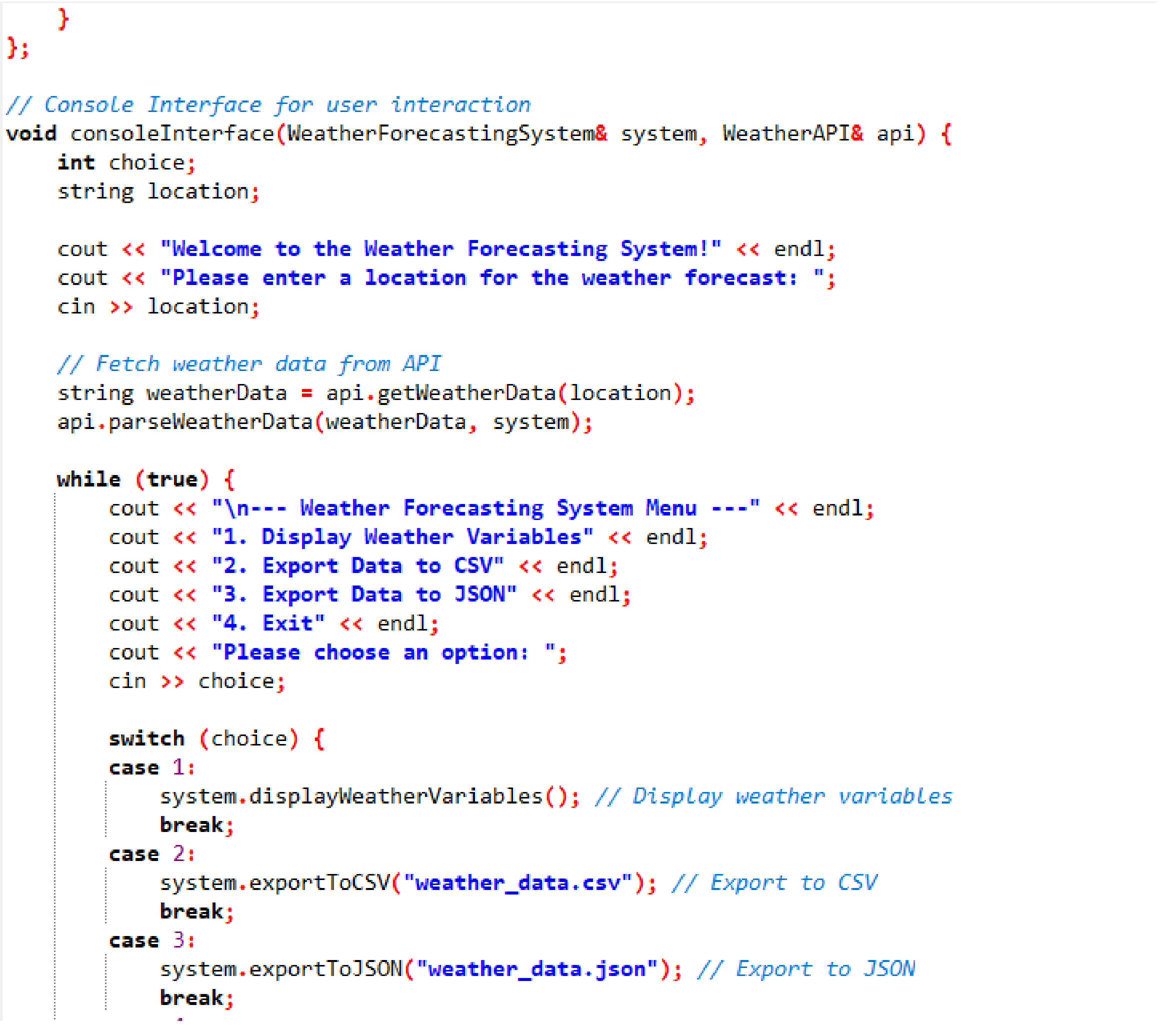
```
// WeatherForecastingSystem Class
class WeatherForecastingSystem {
private:
    vector<WeatherVariable*> weatherVariables;  // Vector to store different weather variables

public:
    // Function to add a WeatherVariable to the system
    void addWeatherVariable(WeatherVariable* var) {
        weatherVariables.push_back(var);
    }

    // Function to display all weather variables
    void displayWeatherVariables() const {
        cout << "Displaying Weather Variables:" << endl;
        for (const auto& var : weatherVariables) {
            var->display(); // Calls the display function of each variable
        }
    }

    // Export weather data to CSV
    void exportToCSV(const string& filename) const {
        ofstream file(filename);
        if (file.is_open()) {
            file << "Variable,Value\n";
            for (const auto& var : weatherVariables) {
                file << var->getName() << "," << var->getValue() << "\n";
            }
            file.close();
            cout << "Data successfully exported to " << filename << endl;
        } else {
            cerr << "Error: Could not open file for writing." << endl;
        }
    }
```



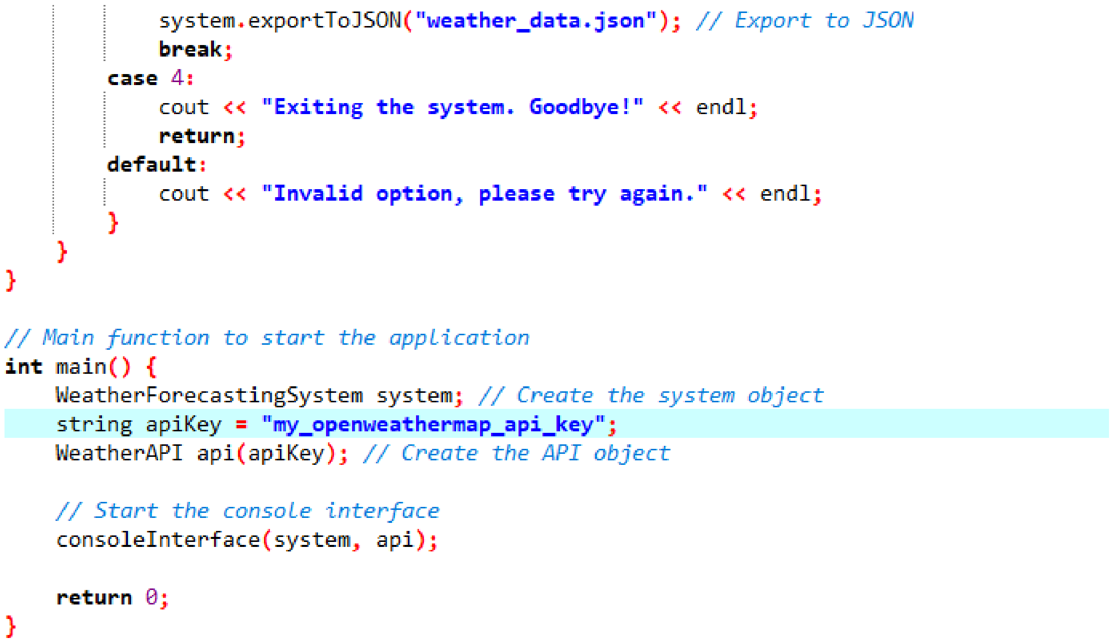
```
    // Export weather data to JSON
    void exportToJSON(const string& filename) const {
        nlohmann::json j;
        for (const auto& var : weatherVariables) {
            j[var->getName()] = var->getValue();
        }
        ofstream file(filename);
        if (file.is_open()) {
            file << j.dump(4); // Pretty-print the JSON with 4 spaces indentation
            file.close();
            cout << "Data successfully exported to " << filename << endl;
        } else {
            cerr << "Error: Could not open file for writing." << endl;
        }
    }

    // Destructor to clean up dynamically allocated memory
    ~WeatherForecastingSystem() {
        for (auto var : weatherVariables) {
            delete var;
        }
    }
};

// WeatherAPI Class for API Integration
class WeatherAPI {
public:
    string apiKey;
    string baseUrl;

    // Constructor initializing the API key and base URL
    WeatherAPI(const string& key) : apiKey(key), baseUrl("http://api.openweathermap.org/data/2.5/weather") {}

    // Function to get weather data from the API
    string getWeatherData(const string& location) {
```



```
    CURL* curl;
    CURLcode res;
    string data;

    curl = curl_easy_init();
    if (curl) {
        string fullUrl = baseUrl + "?q=" + location + "&appid=" + apiKey + "&units=metric";
        curl_easy_setopt(curl, CURLOPT_URL, fullUrl.c_str());

        // Set the callback function to capture the response
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &data);

        res = curl_easy_perform(curl);
        curl_easy_cleanup(curl);
    }
    return data;
}

// Callback function to capture the data returned by the API
static size_t WriteCallback(void* contents, size_t size, size_t nmemb, string* s) {
    s->append((char*)contents, size * nmemb);
    return size * nmemb;
}

// Function to parse the API data and add it to the system
void parseWeatherData(const string& data, WeatherForecastingSystem& system) {
    nlohmann::json jsonData = nlohmann::json::parse(data);
    double temperature = jsonData["main"]["temp"];
    double windSpeed = jsonData["wind"]["speed"];

    // Add parsed weather variables to the system
    system.addWeatherVariable(new Temperature(temperature));
    system.addWeatherVariable(new WindSpeed(windSpeed));
}
```

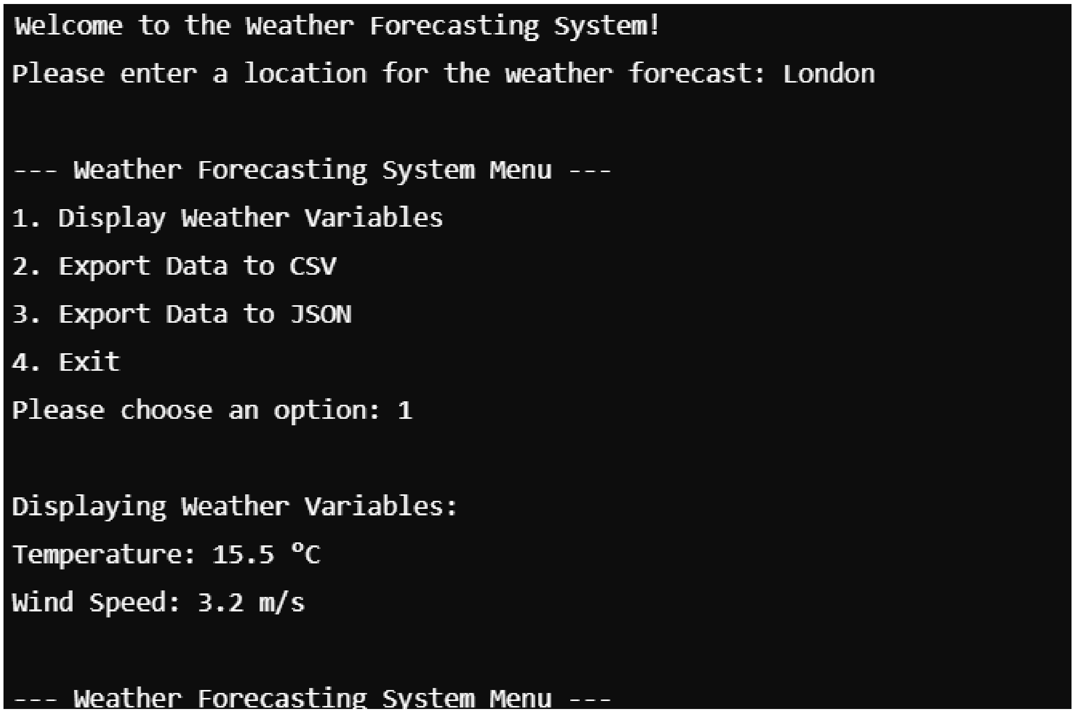


```
    }
};

// Console Interface for user interaction
void consoleInterface(WeatherForecastingSystem& system, WeatherAPI& api) {
    int choice;
    string location;

    cout << "Welcome to the Weather Forecasting System!" << endl;
    cout << "Please enter a location for the weather forecast: ";
    cin >> location;

    // Fetch weather data from API
    string weatherData = api.getWeatherData(location);
    api.parseWeatherData(weatherData, system);

    while (true) {
        cout << "\n--- Weather Forecasting System Menu ---" << endl;
        cout << "1. Display Weather Variables" << endl;
        cout << "2. Export Data to CSV" << endl;
        cout << "3. Export Data to JSON" << endl;
        cout << "4. Exit" << endl;
        cout << "Please choose an option: ";
        cin >> choice;

        switch (choice) {
        case 1:
            system.displayWeatherVariables(); // Display weather variables
            break;
        case 2:
            system.exportToCSV("weather_data.csv"); // Export to CSV
            break;
        case 3:
            system.exportToJSON("weather_data.json"); // Export to JSON
            break;
```

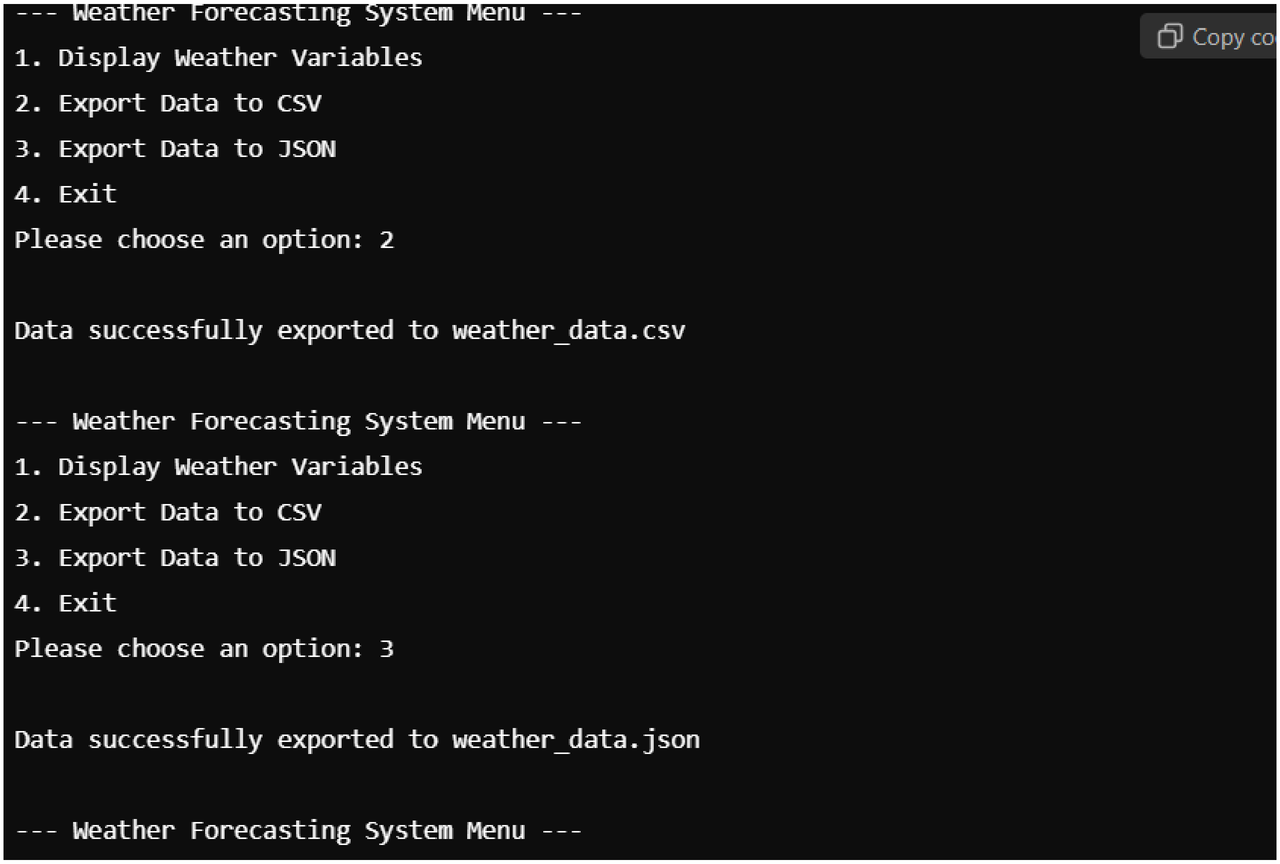


```
            system.exportToJSON("weather_data.json"); // Export to JSON
            break;
        case 4:
            cout << "Exiting the system. Goodbye!" << endl;
            return;
        default:
            cout << "Invalid option, please try again." << endl;
        }
    }
}

// Main function to start the application
int main() {
    WeatherForecastingSystem system; // Create the system object
    string apiKey = "my_openweathermap_api_key";
    WeatherAPI api(apiKey); // Create the API object

    // Start the console interface
    consoleInterface(system, api);

    return 0;
}
```

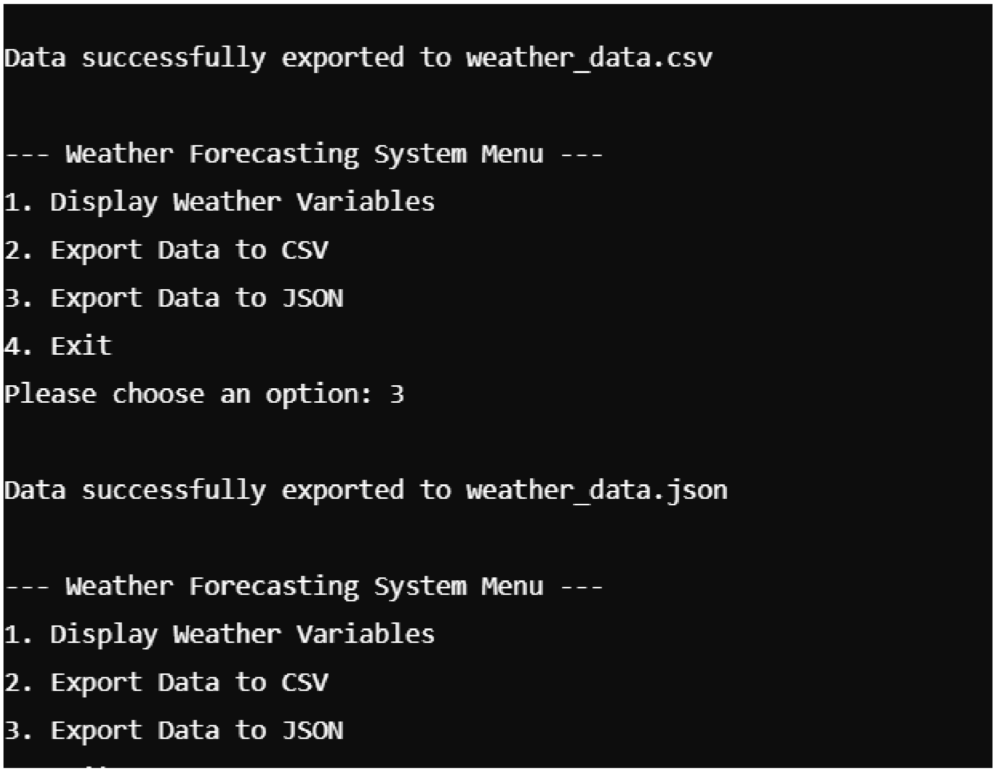
Output:



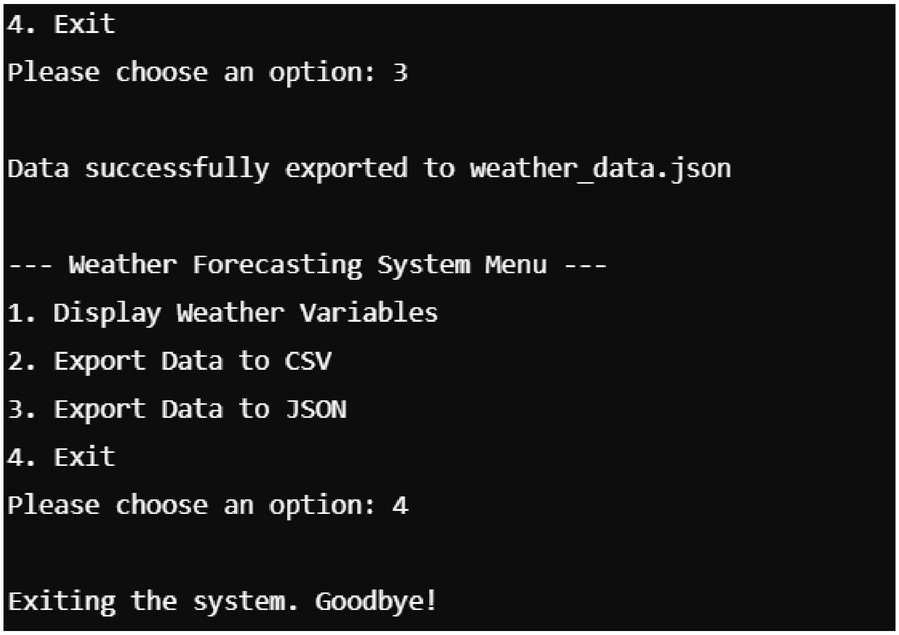
```
Welcome to the Weather Forecasting System!
Please enter a location for the weather forecast: London

--- Weather Forecasting System Menu ---
1. Display Weather Variables
2. Export Data to CSV
3. Export Data to JSON
4. Exit
Please choose an option: 1

Displaying Weather Variables:
Temperature: 15.5 °C
Wind Speed: 3.2 m/s

--- Weather Forecasting System Menu ---
```



```
--- Weather Forecasting System Menu ---
1. Display Weather Variables
2. Export Data to CSV
3. Export Data to JSON
4. Exit
Please choose an option: 2

Data successfully exported to weather_data.csv

--- Weather Forecasting System Menu ---
1. Display Weather Variables
2. Export Data to CSV
3. Export Data to JSON
4. Exit
Please choose an option: 3

Data successfully exported to weather_data.json

--- Weather Forecasting System Menu ---
```



```
Data successfully exported to weather_data.csv

--- Weather Forecasting System Menu ---
1. Display Weather Variables
2. Export Data to CSV
3. Export Data to JSON
4. Exit
Please choose an option: 3

Data successfully exported to weather_data.json

--- Weather Forecasting System Menu ---
1. Display Weather Variables
2. Export Data to CSV
3. Export Data to JSON
```



```
4. Exit
Please choose an option: 3

Data successfully exported to weather_data.json

--- Weather Forecasting System Menu ---
1. Display Weather Variables
2. Export Data to CSV
3. Export Data to JSON
4. Exit
Please choose an option: 4

Exiting the system. Goodbye!
```

Explanation:

Overview

This program is a simple weather forecasting system that fetches weather data from an API, displays it, and allows the user to export it to CSV or JSON files.

Components

1. **WeatherVariable Class**:
   - **Purpose**: Represents a generic weather variable (like temperature or wind speed).
   - **Members**:
     - Name: The name of the weather variable (e.g., "Temperature").
     - value: The value of the weather variable (e.g., 15.5).
   - **Methods**:
     - display(): Prints the name and value.
     - getValue(): Returns the value.
     - getName(): Returns the name.
     - setValue(): Sets a new value.
     - **Destructor**: Cleans up any dynamically allocated memory (though not used here directly).
2. **Derived Classes**:
   - **Temperature**: Inherits from WeatherVariable. It represents temperature and displays it with a "°C" unit.
   - **WindSpeed**: Inherits from WeatherVariable. It represents wind speed and displays it with an "m/s" unit.
3. **WeatherForecastingSystem Class**:

   - **Purpose**: Manages a collection of weather variables and provides functionality to display and export them.

   - **Members**:

     - weatherVariables: A list of pointers to WeatherVariable objects.

   - **Methods**:

     - addWeatherVariable(): Adds a weather variable to the list.
     - displayWeatherVariables(): Displays all weather variables.
     - exportToCSV(): Exports weather variables to a CSV file.
     - exportToJSON(): Exports weather variables to a JSON file.
     - **Destructor**: Cleans up memory by deleting dynamically allocated weather variables.
4. **WeatherAPI Class**:

   - **Purpose**: Handles communication with a weather data API (OpenWeatherMap).
   - **Members**:

     - apiKey: Your API key for accessing the weather data.
     - baseUrl: The base URL for the weather API.

   - **Methods**:

     - getWeatherData(): Makes a network request to fetch weather data for a given location.
     - WriteCallback(): A helper function that processes data received from the API.
     - parseWeatherData(): Parses the JSON response and creates weather variables (Temperature, Wind Speed), adding them to the WeatherForecastingSystem.

5. **consoleInterface Function**:

   - **Purpose**: Provides a simple text-based interface for the user to interact with the system.
   - **Process**:

     - Prompts the user to enter a location.
     - Fetches and parses weather data.
     - Displays a menu to let the user choose to display data, export to CSV or JSON, or exit.
     - Handles user input to perform the selected action.

6. **main Function**:

   - **Purpose**: The entry point of the program.
   - **Process**:

     - Creates an instance of WeatherForecastingSystem.
     - Creates an instance of WeatherAPI with a provided API key.
     - Starts the console interface, allowing user interaction.

Key Points

- **Object-Oriented Design**: The code uses classes to model real-world concepts (weather variables) and their behaviors.
- **Dynamic Memory**: The WeatherForecastingSystem class dynamically allocates memory for weather variables and ensures it is properly cleaned up.
- **API Interaction**: The WeatherAPI class interacts with an external API to fetch weather data.
- **Data Export**: The system can export the collected data to CSV or JSON formats for further use.

This structure keeps the code organized, making it easier to manage and extend.