~\Documents\GitHub\Class-Assignments\EEL3834\City Data Logger\City Data Logger.cpp

```cpp
#include <iostream>
#include <iomanip>
#include <cstring>
#include <cctype>
#include <string>
#include <cstdlib>
#include <cmath>

using namespace std;

// Declares our structure for our Cities
struct City {
    char name[50];
    char code[4];
    double temperatures[7];
    int readingCount;
};

// Allows us to add a city to our data
void addCity(City cities[], int &cityCount){
    bool codeCheck = true;
    City newCity;

    // Loops through taking a name input until our city has a name that is not empty
    do {
        cout << "Enter City Name: ";
        cin.ignore();
        cin.getline(newCity.name, 50);
        if(strlen(newCity.name) == 0){
            cout << "Error: City name cannot be empty. \n";
        }
    } while (strlen(newCity.name) == 0);

    // Makes sure our city code is 3 uppercase letters
    do {
        codeCheck = true;
        cout << "Enter City Code (3 letters): ";
        cin.getline(newCity.code, 4);

        if(strlen(newCity.code) != 3 || isupper(newCity.code[0]) == false || isupper(newCity.code[1]) == false || isupper(newCity.code[2]) == false){
            codeCheck = false;
        }

        // Compares the difference in the our new city code and all existing codes, if the difference is equivalent to zero, the strings are the same and non-unique
        for (int i = 0; i < cityCount && codeCheck; i++){
            if (strcmp(cities[i].code, newCity.code) == 0){
                codeCheck = false;
                break;
            }
        }

        // If the codeCheck returns false it means the code is not unique
        if(codeCheck == false){
            cout << "Error: City code must be unique and exactly 3 letters.\n";
        }
    } while (codeCheck == false);

    // Keeps track of how many temperature inputs we have
    int tempTrack = 0;
    while (true) {
        cout << "Enter number of temperature readings (max 7): ";
        cin >> tempTrack;
        // If we have a request to input a number of temperatures outside our range of 1 to 7, we return an error
        if (tempTrack > 7 || tempTrack < 1) {
            cout << "Error: You must enter between 1 and 7 temperatures. \n";
        }
        else {
            break;
        }
    }

    // Stores how many readings this city has
    newCity.readingCount = tempTrack;

    // Iterates through storing our temperature data a number of times equal to the number above
    for(int i = 0; i < tempTrack; i++){

        cout << "Enter temperature " << (i+1) << ": ";
        while (true){
            cin >> newCity.temperatures[i];
            // Loops through until a temperature between -100 and 100 degrees C is given
            if (newCity.temperatures[i] > 100 || newCity.temperatures[i] < -100){
                cout << "Error: Temperature must be between -100 and 100 degrees C. \n";
            }
            else {
                break;
            }
        }

    }

    // Replaces the data storage for cities at our cityCount point, overwriting/adding new data
    cities[cityCount] = newCity;
    cout << "City data added successfully!\n";
    cout << endl;
    // Iterates to allow new data to be stored when reused
    cityCount++;
    return;
}

// Calculates the average temperature across all cities combined
double calculateAverageTemp(City cities[], int cityCount) {
    double sum = 0.0;
    int totalTemps = 0;

    // Iterates through temperatures within cities, cycling through a cities temperatures then moving to the next city
    for (int i = 0; i < cityCount; i++) {
        for (int j = 0; j < cities[i].readingCount; j++) {
            // As the program cycles through temperatures, it sums together the temperature data and increments the total temperatures logged
            sum += cities[i].temperatures[j];
            totalTemps++;
        }
    }

    // Prevents division by 0 if no data is logged
    if (totalTemps == 0){
        return(0.00);
    }

    // Since return does not seem to work with setprecision, we round our return value to two places "manually"
    return(round((sum / totalTemps) * 100) / 100);
}

// Calculates the average temperature for a single city by being passed its temperatures array and length
double calculateAverageTemp(double temps[], int count) {
    // Prevents division by 0 again by returning 0 if there is no data written
    if (count == 0) return 0.00;

    double sum = 0.00;
    // Sums together all of the temperatures
    for (int i = 0; i < count; i++) {
        sum = temps[i] + sum;
    }
    // Rounds the average to two decimal places
    return round((sum / count) * 100) / 100;
}

// Finds the hottest and coldest cities in our data given the cities array and the amount of cities
void findHottestAndColdest(City cities[], int cityCount) {
    if (cityCount == 0) {
        cout << "No cities available.\n";
        return;
    }

    int hottestCount = 0;
    int coldestCount = 0;
    // Passes the first average through our calculateAverageTemp function to give us a baseline to compare other temperature averages to
    double hottestAvg = calculateAverageTemp(cities[0].temperatures, cities[0].readingCount);
    double coldestAvg = hottestAvg;

    // Iterates through all data to find the min and max temperature averages of all cities
    for (int i = 1; i < cityCount; i++) {
        double avg = calculateAverageTemp(cities[i].temperatures, cities[i].readingCount);
        // Compares our average to the hottest/coldest data and replaces it when necessary, saves index to print out below
        if (avg > hottestAvg) {
            hottestAvg = avg;
            hottestCount = i;
        }
        if (avg < coldestAvg) {
            coldestAvg = avg;
            coldestCount = i;
        }
    }

    cout << "Hottest city: " << cities[hottestCount].name << " (" << cities[hottestCount].code
         << ") Average: " << hottestAvg << " C\n";
    cout << "Coldest city: " << cities[coldestCount].name << " (" << cities[coldestCount].code
         << ") Average: " << coldestAvg << " C\n\n";
}

// Searches for a city to find its respective temperatures, code, and name
void searchCity(City cities[], int cityCount) {
    if (cityCount == 0) {
        cout << "No cities available to search.\n\n";
        return;
    }
    int choice = 0;

    while(true){
    cout << "Search by (1 for Code, 2 for Name): ";
    cin >> choice;

    if(choice == 1 || choice == 2){
        break;
    } else {
        cout << "Invalid input!";
    }
    }
    double avg = 0;
    if (choice == 1){
        char searchCode[4];
        cin.ignore(); // We use cin.ignore so that the getline command does not take the empty line as an input
        cout << "Enter City Code: ";
        cin.getline(searchCode, 4);

        bool found = false;
        for (int i = 0; i < cityCount; i++) {
            if (strcmp(cities[i].code, searchCode) == 0) { // If there is no difference in the given code and current indexes code, we list the data at the city
                cout << "\nCity found:\n\n";
                cout << "Name: " << cities[i].name << "\n";
                cout << "Code: " << cities[i].code << "\n";
                cout << "Temperatures: ";
                for (int j = 0; j < cities[i].readingCount; j++) { // Iterates through the temperatures within the city (array in array)
                    cout << fixed << setprecision(1) << cities[i].temperatures[j]; // Changed setprecision based on example
                    avg = avg + cities[i].temperatures[j];
                    if (j < cities[i].readingCount - 1) cout << ", ";
                }
                cout << "]\n";
                cout << "Average: " << fixed << setprecision(2) << avg/cities[i].readingCount << " C \n\n"; // Used set precision to find the average temperature to two decimal places
                found = true;
                break;
            }
        }
        if (found == false) {
        cout << "Error: No city found with code " << searchCode << ".\n\n";
    }
    }
    // This option searches by the name instead of city code
    else if (choice == 2){

        char searchName[50];
        cin.ignore(); // Used cin.ignore to prevent grabbing empty line
        cout << "Enter City Name (or press enter for overall average):";
        cin.getline(searchName, 50);

        bool found = false;
        for (int i = 0; i < cityCount; i++) {
            if (strcmp(cities[i].name, searchName) == 0) {
                cout << "\nCity Found:\n\n";
                cout << "Code: " << cities[i].code << "\n";
                cout << "Name: " << cities[i].name << "\n";
                cout << "Temperature Readings: [";
                for (int j = 0; j < cities[i].readingCount; j++) { // Iterates through the temperatures within the city (array in array)
                    cout << fixed << setprecision(1) << cities[i].temperatures[j];
                    avg = avg + cities[i].temperatures[j];
                    if (j < cities[i].readingCount - 1) cout << ", ";
                }
                cout << "]\n";
                cout << "Average: " << fixed << setprecision(2) << avg/cities[i].readingCount << " C \n\n";
                found = true;
                break;
            }
        }
        if (found == false) {
        cout << "Error: No city found with name " << searchName << ".\n\n";
    }
    }
}

// Allows us to remove cities from our data
void removeCity(City cities[], int &cityCount) {
    if (cityCount == 0) {
        cout << "No cities to remove.\n\n";
        return;
    }

    char codeToRemove[4];
    cin.ignore();
    cout << "Enter City Code to remove: ";
    cin.getline(codeToRemove, 4);

    bool found = false;
    for (int i = 0; i < cityCount; i++) { // If
        if (strcmp(cities[i].code, codeToRemove) == 0) {
            for (int j = i; j < cityCount - 1; j++) { // Finds where the city to be removed's index is located
                cities[j] = cities[j + 1]; // Moves existing cities over to fill in empty space
            }
            cityCount--;
            found = true;
            cout << "City " << codeToRemove << " removed successfully!\n\n";
            break;
        }
    }

    if (found == false) {
        cout << "Error: No city found with code " << codeToRemove << ".\n\n";
    }
}

// Holds our menu and initializes variables
int main()
{
        cout << fixed << setprecision(2); // Sets precision globally
        City cities[100]; // Lets us have up to 100 cities
        int cityCount = 0; // Says we have 0 cities to start

    while (true) {

        int choice = 0;

        cout << "===== Weather Data Logger =====" << endl;
        cout << "1. Add City and Temperature Readings" << endl;
        cout << "2. Calculate Average Temperature" << endl;
        cout << "3. Find Hottest and Coldest Cities" << endl;
        cout << "4. Search City" << endl;
        cout << "5. Remove City Data" << endl;
        cout << "6. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;
        cout << endl;

        if (choice == 1) {

            addCity(cities, cityCount);
        }
        else if (choice == 2){

            cin.ignore();
            char input[4]; // Takes a city code
            cout << "Enter City Code (or press enter for overall average): ";
            cin.getline(input, 4);

            if (strlen(input) == 0) { // If nothing is inputted, uses our calculateAverageTemp function to find average temp of all cities
                double avg = calculateAverageTemp(cities, cityCount);
                cout << "Average temperature across all cities: " << avg << " C\n\n";
            } else {
                bool found = false;
                for (int i = 0; i < cityCount; i++) { // Iterates through cities to find matching code
                    if (strcmp(cities[i].code, input) == 0) { // Compares input to existing code, if there is no difference we find the average temperature of that location
                        double avg = calculateAverageTemp(cities[i].temperatures, cities[i].readingCount);
                        cout << "Average temperature for " << input << ": " << avg << " C\n\n";
                        found = true;
                        break;
                    }
                }
                if (found == false) {
                    cout << "Error: No city found with code " << input << ".\n\n";
                }
            }

        }
        else if (choice == 3){
            findHottestAndColdest(cities, cityCount);
        }
        else if (choice == 4){
            searchCity(cities, cityCount);
        }
        else if (choice == 5){
            removeCity(cities, cityCount);
        }
        else if (choice == 6){
            cout << "Exiting the program... Goodbye!";
            exit(0);
        }
    }
}
```

```python
# Function used to add cities and their temperatures to our data
def addCity(cities):

    cityName = ""
    cityCode = ""

    # Checks if the name inputted is valid by making sure the value is not empty (0 characters long)
    validName = False
    while not validName:
        cityName = input("Enter City Name: ")

        if len(cityName) == 0:
            print("Error: City name cannot be empty.")
        else:
            validName = True

    # Checks that the inputted code is 3 characters long and only letters.
    validCode = False
    while not validCode:
        cityCode = input("Enter City Code: ")

        if len(cityCode) != 3 or not cityCode.isalpha():
            print("Error: City code must be unique and exactly 3 letters.")
        else:
            # Iterates through existing codes to find a match, if it does the code is invalid
            duplicateCode = False
            for city in cities:
                if city['code'] == cityCode.upper():
                    print("Error: City code must be unique and exactly 3 letters.")
                    duplicateCode = True
            if not duplicateCode:
                validCode = True
    # Moves on to checking temperature inputs, makes sure we are taking 1 to 7 inputs (ensure it is an integer)
    while True:
        try:
            tempCount = int(input("Enter number of temperature readings (max 7): "))
            if 1 <= tempCount <= 7:
                break
            else:
                print("Error: You must enter between 1 and 7 temperatures.")
        except ValueError:
            print("Error: You must enter between 1 and 7 temperatures.")

    # Initializes the list for city temperatures, takes as many temperature readings as listed above
    cityTemperatures = []
    for i in range(tempCount):
        while True:
            try:
                # Makes sure the temperature comes in form of a float, and checks to make sure the temperature
                # is within range
                temp = float(input(f"Enter temperature {(i+1)}: "))
                if -100 <= temp <= 100:
                    cityTemperatures.append(temp)
                    break
                else:
                    print("Error: Temperature must be between -100 and 100 degrees.")
            except ValueError:
                print("Error: Temperature must be between -100 and 100 degrees.")


    # Initializes our city as a dictionary
    city = {
        'name': cityName,
        'code': cityCode.upper(),
        'temperatures': cityTemperatures
    }

    # Adds our data to the storage by adding to the end of our cities list
    cities.append(city)
    print("City data added successfully!\n")

# Function to calculate the average temperature of a city or all cities
def calculateAverageTemperature(cities):

    tempSum = 0
    tempCount = 0
    choice = input("Enter City Code (or press enter for overall average): ")

    # If we get an empty input, we check all cities average temperature
    if len(choice) == 0:

        print("Calculating overall average temperature...")
        # Iterates through all cities, and all temperatures within each city to find total temp and amount of temps
        for city in cities:
            for temp in city['temperatures']:
                tempSum += temp
                tempCount += 1
        averageTemp = tempSum / tempCount
        print("Average temperature across all cities: ", round(averageTemp, 2), "C \n")

    # If we get a length of 3, we know it is a code, and check to see if it matches an existing code
    elif len(choice) == 3:
        exists = False
        # Checks through each city and checks their code to see if it matches
        # if it does it finds the average temperature
        for city in cities:
            if city['code'].upper() == choice.upper():
                exists = True
                for temp in city['temperatures']:
                    tempSum += temp
                    tempCount += 1
                # If there are no temperatures we just output 0.00 C to prevent division by zero
                if tempCount == 0:
                    print("Average temperature for", city['code'] + ":", 0.00, "C\n")
                # Otherwise we take our average
                else:
                    print("Average temperature for", city['code'] + ":", "{:.2f}".format(tempSum / tempCount), "C\n")
                break

        # If a matching city cannot be located, we send back an error
        if not exists:
            print("Error: No city found.")

# Finds the hottest and coldest cities in all of our data
def hottestAndColdestCities(cities):

    # If there are no cities, we cannot find the hottest or coldest
    if len(cities) == 0:
        print("Error: No cities found.\n")
        return

    hottestCity = ""
    coldestCity = ""
    hottestCityTemp = 0
    coldestCityTemp = 0

    # We iterate through all cities and take the sum and count of temperature data, and find the average
    for city in cities:
        temps = city['temperatures']

        if len(temps) == 0:
            print("Error: No temperatures found.\n")
            continue

        averageCityTemp = sum(temps)/len(temps)

        # If the current city is hotter OR if the hottest city is empty we replace it with our current data
        if averageCityTemp > hottestCityTemp or hottestCity == "":
            hottestCity = city['name']
            hottestCityTemp = averageCityTemp
        # If the current city is colder OR if the coldest city is empty we replace it with our current data
        if averageCityTemp < coldestCityTemp or coldestCity == "":
            coldestCity = city['name']
            coldestCityTemp = averageCityTemp

    # Prints our hottest and coldest city averages to two decimal places
    print(f"Hottest city: {hottestCity} (Average: {hottestCityTemp:.2f} C)")
    print(f"Coldest city: {coldestCity} (Average: {coldestCityTemp:.2f} C)")
    print()

# Find information for a single city based on search by code or name
def searchCity(cities):

    # Loops until given either 1 or 2
    while True:
        try:
            choice = int(input("Search by (1 for Code, 2 for Name): "))
            if choice == 1 or choice == 2:
                break

        except ValueError:
            print("Error: Please enter 1 or 2.")

    exists = False

    # If they choose to search by code they enter the code and it is compared to all codes
    # If a match is found it displays all data given on that city
    if choice == 1:
        codeTrack = input("Enter City Code: ")
        print()

        for city in cities:
            if city['code'].upper() == codeTrack.upper():
                exists = True
                print("City Found:\n")
                print("Code:", city['code'])
                print("Name:",city['name'])
                print("Temperature Readings:",city['temperatures'])
                averageTemp = sum(city['temperatures']) / len(city['temperatures'])
                print(f"Average: {averageTemp:.2f} C")
                print("\n")
        if exists == False:
            print("Error: No city found with code " + codeTrack.upper() + ".\n")

    # If they choose to search by name they enter the name and it is compared to all names
    # If a match is found it displays all data given on that city
    if choice == 2:
        nameTrack = input("Enter City Name: ")
        print()

        for city in cities:
            if city['name'].upper() == nameTrack.upper():
                exists = True
                print("City Found:\n")
                print("Code:", city['code'])
                print("Name:",city['name'])
                print("Temperature Readings:",city['temperatures'])
                averageTemp = sum(city['temperatures']) / len(city['temperatures'])
                print(f"Average: {averageTemp:.2f} C")
                print("\n")
        if exists == False:
            print("Error: No city found with code " + nameTrack.upper() + ".\n")

# Removes cities and their temperatures from our data
def removeCity(cities):

    # Takes a city code and deletes the matching city code in our data
    deletedCityCode = input("Enter City Code to remove: ")
    exists = False

    for city in cities:
        if city['code'].upper() == deletedCityCode.upper():
            cities.remove(city) # Finds the city within the cities and removes it from its current location
            exists = True
            print("City", deletedCityCode,"removed successfully!")

    if exists == False:
        print("Error: No city found.\n")

    print()

# Holds our menu and choices to run other functions, always runs until exited
def main():
    # Initializes an empty cities list upon opening
    cities = []

    while True:
        print("===== Weather Data Logger =====")
        print("1. Add City and Temperature Readings")
        print("2. Calculate Average Temperature")
        print("3. Find Hottest and Coldest Cities")
        print("4. Search City")
        print("5. Remove City Data")
        print("6. Exit")

        choice = int(input("Enter your Choice: "))
        print()

        if choice == 1:
            addCity(cities)
        elif choice == 2:
            calculateAverageTemperature(cities)
        elif choice == 3:
            hottestAndColdestCities(cities)
        elif choice == 4:
            searchCity(cities)
        elif choice == 5:
            removeCity(cities)
        elif choice == 6:

            print()
            print("Exiting the program... Goodbye!")
            exit()
        else:

            print("Invalid ")

main()
```