File - C:\Users\nicho\Desktop\GitHub\CS5008_LastAssignment\AirQualityProcessor\final_project\Date.h

```
 1
 2 #ifndef FINAL_PROJECT_DATE_H
 3 #define FINAL_PROJECT_DATE_H
 4 #include "ostream"
 5
 6 using namespace std;
 7
 8 class Date
 9 {
10     friend ostream& operator << (ostream&, const Date&); // method overloading
11     friend istream& operator >> (istream&, Date&);
12
13 public:
14
15     // default constructor
16     Date();
17
18     // specific constructor
19     Date(int d, int m, int y);
20
21     // destructor
22     ~Date();
23
24     // setters
25     void setDay(int d);
26     void setMonth(int m);
27     void setYear(int y);
28     void setDmy(int d, int m, int y);
29
30     // getters
31     int getDay() const;
32     int getMonth() const;
33     int getYear() const;
34
35     // printf
36     void print() const;
37
38     // Overloading operators
39     bool operator<(const Date& otherDate) const;
40     bool operator>(const Date& otherDate) const;
41     bool operator==(const Date& otherDate) const;
42     bool operator<=(const Date& otherDate) const;
43     bool operator>=(const Date& otherDate) const;
```

```
44      bool operator!=(const Date& otherDate) const;
45
46      // Overloading math operations
47      Date operator++();
48      Date operator++(int);
49      Date operator--();
50      Date operator--(int);
51
52  private:
53
54      // data members
55      int day;
56      int month;
57      int year;
58  };
59
60  #endif
61
```

File - C:\Users\nicho\Desktop\GitHub\CS5008_LastAssignment\AirQualityProcessor\final_project\Time.h

```
1  //
2  // Created by hycwy on 4/9/2022.
3  //
4
5  #ifndef FINAL_PROJECT_TIME_H
6  #define FINAL_PROJECT_TIME_H
7  #include "ostream"
8
9  using namespace std;
10
11 class Time {
12
13     friend ostream& operator << (ostream&, const Time&); // method overloading
14     friend istream& operator >> (istream&, Time&);
15
16 public:
17
18     // default constructor
19     Time();
20
21     // specific constructor
22     Time(int s, int m, int h);
23
24     // destructor
25     ~Time();
26
27     // setters
28     void setSec(int s);
29     void setMin(int m);
30     void setHour(int h);
31     void setSmh(int s, int m, int h);
32
33     // getters
34     int getSec() const;
35     int getMin() const;
36     int getHour() const;
37
38     // printf
39     void print() const;
40
41     // Overloading operators
42     bool operator<(const Time& otherTime) const;
43     bool operator>(const Time& otherTime) const;
```

File - C:\Users\nicho\Desktop\GitHub\CS5008_LastAssignment\AirQualityProcessor\final_project\Time.h

```
44      bool operator==(const Time& otherTime) const;
45      bool operator<=(const Time& otherTime) const;
46      bool operator>=(const Time& otherTime) const;
47      bool operator!=(const Time& otherTime) const;
48
49      // Overloading math operations
50      Time operator++();
51      Time operator++(int);
52      Time operator--();
53      Time operator--(int);
54
55  private:
56
57      // data members
58      int sec;
59      int min;
60      int hour;
61  };
62
63
64  #endif //FINAL_PROJECT_TIME_H
65
```

File - C:\Users\nicho\Desktop\GitHub\CS5008_LastAssignment\AirQualityProcessor\final_project\Date.cpp

```cpp
#include <iostream>
#include "Date.h"

using namespace std;

// default constructor
Date::Date()
{
    day = 1;
    month = 1;
    year = 2000;
}

// specific constructor
Date::Date(int d, int m, int y)
{
    day = d;
    month = m;
    year = y;
}


// destructor
Date::~Date() {

}

// setters
void Date::setDay(int d)
{
    day = d;
}

void Date::setMonth(int m)
{
    month = m;
}

void Date::setYear(int y)
{
    year = y;
}

```

```
44 void Date::setDmy(int d, int m, int y)
45 {
46     day = d;
47     month = m;
48     year = y;
49 }
50
51 // getters
52 int Date::getDay() const
53 {
54     return day;
55 }
56
57 int Date::getMonth() const
58 {
59     return month;
60 }
61
62 int Date::getYear() const
63 {
64     return year;
65 }
66
67 // printf
68 void Date::print() const
69 {
70     cout << day << "/"
71          << month << "/"
72          << year;
73 }
74
75 // ostream operator
76 ostream& operator << (ostream& osObject, const Date& date1) //creating an instance of a date class, i.e. date1
77 {
78     osObject << date1.day
79              << "/" << date1.month
80              << "/" << date1.year;
81     return osObject;
82 }
83
84 // istream operator
85 istream& operator >> (istream& isObject, Date& date1)
86 {
```

```
 87     isObject >> date1.day >> date1.month >> date1.year;
 88     return isObject;
 89 }
 90
 91 // Overloading "equal to" operator
 92 bool Date::operator==(const Date& otherDate) const
 93 {
 94     if (day == otherDate.day && month == otherDate.month
 95         && year == otherDate.year)
 96         return true;
 97     else
 98         return false;
 99 }
100
101 // Overloading "not equal to" operator
102 bool Date::operator!=(const Date& otherDate) const
103 {
104     return !(*this == otherDate);
105 }
106
107 // Overloading "less than or equal to" operator
108 bool Date::operator<=(const Date& otherDate) const
109 {
110     return (*this < otherDate || *this == otherDate);
111 }
112
113 // Overloading "less than" operator
114 bool Date::operator<(const Date& otherDate) const
115 {
116     if ((year < otherDate.year) ||
117         (year == otherDate.year && month < otherDate.month) ||
118         (year == otherDate.year && month == otherDate.month && day < otherDate.day))
119         return true;
120     else
121         return false;
122 }
123
124 // Overloading "greater than or equal to" operator
125 bool Date::operator>=(const Date& otherDate) const
126 {
127     return !(*this < otherDate);
128 }
129
```

```
130 // Overloading "greater than" operator
131 bool Date::operator>(const Date& otherDate) const
132 {
133     return !(*this <= otherDate);
134 }
```

File - C:\Users\nicho\Desktop\GitHub\CS5008_LastAssignment\AirQualityProcessor\final_project\main.cpp

```cpp
#include <iostream>
#include <fstream>
#include <iomanip>
#include <algorithm>
#include "vector"
#include "string"
#include "Date.h"
#include "Date.cpp"
#include "Time.h"
#include "Time.cpp"
#include "AirQuality.h"
#include "AirQuality.cpp"
using namespace std;

void getData(ifstream &inFile, vector<string>& fields, vector<AirQuality>& airQualityCollection);
Date processDate(string inputStr);
Time processTime(string inputStr);
double calculateAvgTemp(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection);
double calculateRelHumidity(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection);
double calculateAbsHumidity(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection);
void displayTempAndRelHumidity(Date dateInput, Time timeInput, double &currentTemp, double &currentRelHumidity, vector<AirQuality>&
airQualityCollection);
double displayMaxTemp(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection);
double displayMaxRelHumidity(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection);
double displayMaxAbsHumidity(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection);
void findTempHigherThanAvg(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection, vector<AirQuality>& tempHigherThanAvg);
void findRelHumidHigherThanAvg(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection, vector<AirQuality>&
relHumidHigherThanAvg);
void findAbsHumidHigherThanAvg(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection, vector<AirQuality>&
absHumidHigherThanAvg);
void showWelcomeMsg();
void showMenu();
int  userChoice();
bool validMonth(int monthInput);
void doUserChoice(int option, vector<AirQuality>& airQualityCollection);
Time getValidTime();
int getValidInputMonth();
int getValidInputYear();
int getValidInputDay(int inputYear, int inputMonth);
void doCalOrDis(int option, int yearInput, int monthInput,vector<AirQuality>& airQualityCollection);
void doCalThanAvg(int option, int yearInput, int monthInput, vector<AirQuality>& airQualityCollection, vector<AirQuality>&
absHumidHigherThanAvg);
void doFindTempAndHum(Date inputDate, Time inputTime, double &currentTemp, double &currentRelHumidity, vector<AirQuality>&
```

File - C:\Users\nicho\Desktop\GitHub\CS5008_LastAssignment\AirQualityProcessor\final_project\main.cpp

```
39 airQualityCollection);
40
41
42 int main()
43 {
44     ifstream myFile("AirQualityUCI.csv");
45
46     if (!myFile.is_open())
47     {
48         cout << "File not found." << endl;
49         return 1;
50     }
51
52     vector<string> fields;
53     vector<AirQuality> airQualityCollection;
54     getData(myFile, fields, airQualityCollection);
55     showWelcomeMsg();
56     showMenu();
57     int res = userChoice();
58     int EXIT = -1;
59     while (res != EXIT)
60     {
61         doUserChoice(res, airQualityCollection);
62         cout << endl;
63         showMenu();
64         res = userChoice();
65         if (res == EXIT)
66         {
67             break;
68         }
69     }
70     return 0;
71 }
72
73 /**
74  * Read and process data from csv file
75  * @param inFile input stream to operate on files
76  * @param fields vector stores each line info
77  * @param airQualityCollection vector stores AirQuality objects
78  */
79 void getData(ifstream &inFile, vector<string>& fields, vector<AirQuality>& airQualityCollection)
80 {
81     static const int DATE_IDX = 0;
```

```cpp
    static const int TIME_IDX = 1;
    static const int TEMP_IDX = 12;
    static const int REL_HUMID_IDX = 13;
    static const int ABS_HUMID_IDX = 14;

    string strs;
    string delimiter = ",";
    size_t pos;
    // skip the first line
    getline(inFile, strs);
    // read each line, store as strs
    while (getline(inFile, strs))
    {
        size_t size = strs.size() - 1;
        // skip empty line (',,,,')
        if (strs[0] == ',')
        {
            break;
        }
        for (int i = 0; i < size; ++i)
        {
            pos = strs.find(delimiter, i);
            if (pos < size)
            {
                string token = strs.substr(i, pos - i);
                fields.push_back(token);
                i = pos + delimiter.size() - 1;
            }
        }

        // create date object with element in fields vector (index = 0)
        Date dateVal = processDate(fields.at(DATE_IDX));
        // create time object with element in fields vector (index = 1)
        Time timeVal = processTime(fields.at(TIME_IDX));
        // convert other parameters from string to double
        double temperature = stod(fields.at(TEMP_IDX));
        double relHumid = stod(fields.at(REL_HUMID_IDX));
        double absHumid = stod(fields.at(ABS_HUMID_IDX));
        // create AirQuality object, store in airQualityCollection vector
        AirQuality airQualityVal(dateVal, timeVal, temperature, relHumid, absHumid);
        airQualityCollection.push_back(airQualityVal);
        fields.clear();
    }
```

```
125
126     inFile.close();
127 }
128
129 /**
130  * Convert string to Date object
131  * @param inputStr input string of Date info
132  * @return Date object
133  */
134 Date processDate(string inputStr)
135 {
136     string delimiter = "/";
137     size_t pos;
138     string strs = inputStr + delimiter;
139     size_t size = strs.size();
140     vector<int> dateValues;
141     // split the input string to 3 substrings, convert to int
142     for (int i = 0; i < size; ++i)
143     {
144         pos = strs.find(delimiter, i);
145         if (pos < size)
146         {
147             string token = strs.substr(i, pos - i);
148             int tokenNum = stoi(token);
149             dateValues.push_back(tokenNum);
150             i = pos + delimiter.size() - 1;
151         }
152     }
153
154     static const int DAY_IDX = 1;
155     static const int MONTH_IDX = 0;
156     static const int YEAR_IDX = 2;
157     // create date object with elements in dateValues vector
158     Date dateVal(dateValues.at(DAY_IDX), dateValues.at(MONTH_IDX), dateValues.at(YEAR_IDX));
159     return dateVal;
160 }
161
162 /**
163  * Convert string to Time object
164  * @param inputStr input string of Time info
165  * @return Time object
166  */
167 Time processTime(string inputStr)
```

```
168 {
169     string delimiter = ":";
170     size_t pos;
171     string strs = inputStr + delimiter;
172     size_t size = strs.size();
173     vector<int> timeValues;
174     // split the input string to 3 substrings, convert to int
175     for (int i = 0; i < size; ++i)
176     {
177         pos = strs.find(delimiter, i);
178         if (pos < size)
179         {
180             string token = strs.substr(i, pos - i);
181             int tokenNum = stoi(token);
182             timeValues.push_back(tokenNum);
183             i = pos + delimiter.size() - 1;
184         }
185     }
186 
187     static const int HOUR_IDX = 0;
188     static const int MIN_IDX = 1;
189     static const int SEC_IDX = 2;
190     // create Time object with elements in timeVal vector
191     Time timeVal(timeValues.at(SEC_IDX), timeValues.at(MIN_IDX), timeValues.at(HOUR_IDX));
192     return timeVal;
193 }
194 
195 /**
196  * Calculate average temperature for that month
197  * @param yearInput target year
198  * @param monthInput target month
199  * @param airQualityCollection vector stores AirQuality objects
200  * @return average temperature for that month
201  */
202 double calculateAvgTemp(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection)
203 {
204     static const double MISS_VALUE = -200;
205     double totalTemp = 0;
206     int count = 0;
207     for (int i = 0; i < airQualityCollection.size(); i++){
208         if (airQualityCollection[i].getDate().getYear() == yearInput && airQualityCollection[i].getDate().getMonth() == monthInput){
209             if (airQualityCollection[i].getTemp() != MISS_VALUE){
210                 totalTemp += airQualityCollection[i].getTemp();
```

```
211                 count++;
212             }
213         }
214     }
215     double avgTemp = totalTemp / count;
216 
217     if (count == 0){
218         return 0;
219     }
220     else{
221         return avgTemp;
222     }
223 }
224 
225 /**
226   * Calculate average relative humidity for that month
227   * @param yearInput target year
228   * @param monthInput target month
229   * @param airQualityCollection vector stores AirQuality objects
230   * @return average relative humidity for that month
231   */
232 double calculateRelHumidity(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection)
233 {
234     static const double MISS_VALUE = -200;
235     double totalHumidity = 0;
236     int count = 0;
237     for (int i = 0; i < airQualityCollection.size(); i++){
238         if (airQualityCollection[i].getDate().getYear() == yearInput && airQualityCollection[i].getDate().getMonth() == monthInput){
239             if (airQualityCollection[i].getRelativeHumidity() != MISS_VALUE){
240                 totalHumidity += airQualityCollection[i].getRelativeHumidity();
241                 count++;
242             }
243         }
244     }
245     double avgHumidity = totalHumidity / count;
246 
247     if (count == 0){
248         return 0;
249     }
250     else{
251         return avgHumidity;
252     }
253 }
```

```cpp

/**
 * Calculate average absolute humidity for that month
 * @param yearInput target year
 * @param monthInput target month
 * @param airQualityCollection vector stores AirQuality objects
 * @return average absolute humidity for that month
 */
double calculateAbsHumidity(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection)
{
    static const double MISS_VALUE = -200;
    double totalHumidity = 0;
    int count = 0;
    for (int i = 0; i < airQualityCollection.size(); i++){
        if (airQualityCollection[i].getDate().getYear() == yearInput && airQualityCollection[i].getDate().getMonth() == monthInput){
            if (airQualityCollection[i].getAbsHumidity() != MISS_VALUE){
                totalHumidity += airQualityCollection[i].getAbsHumidity();
                count++;
            }
        }
    }
    double avgHumidity = totalHumidity / count;

    if (count == 0){
        return 0;
    }
    else{
        return avgHumidity;
    }
}

/**
 * Find the temperature, and relative humidity at that date and time
 * @param dateInput target date
 * @param timeInput target time
 * @param currentTemp temperature at that date and time
 * @param currentRelHumidity relative humidity at that date and time
 * @param airQualityCollection vector stores AirQuality objects
 */
void displayTempAndRelHumidity(Date dateInput, Time timeInput, double &currentTemp, double &currentRelHumidity, vector<AirQuality>&
airQualityCollection)
{
    for (int i = 0; i < airQualityCollection.size(); i++){
```

```
296         if (dateInput == airQualityCollection[i].getDate() && timeInput == airQualityCollection[i].getTime()){
297             currentTemp = airQualityCollection[i].getTemp();
298             currentRelHumidity = airQualityCollection[i].getRelativeHumidity();
299             return;
300         }
301     }
302     currentTemp = 0;
303     currentRelHumidity = 0;
304 }
305 
306 /**
307  * Find the highest temperature for that month
308  * @param yearInput target year
309  * @param monthInput target month
310  * @param airQualityCollection vector stores AirQuality objects
311  * @return highest temperature for that month
312  */
313 double displayMaxTemp(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection)
314 {
315     static const double THRESHOLD = -100;
316     double maxTemp = THRESHOLD;
317     for (int i = 0; i < airQualityCollection.size(); i++){
318         if (airQualityCollection[i].getDate().getYear() == yearInput && airQualityCollection[i].getDate().getMonth() == monthInput) {
319             maxTemp = max(maxTemp, airQualityCollection[i].getTemp());
320         }
321     }
322     return maxTemp;
323 }
324 
325 /**
326  * Find the highest relative humidity for that month
327  * @param yearInput target year
328  * @param monthInput target month
329  * @param airQualityCollection vector stores AirQuality objects
330  * @return highest relative humidity for that month
331  */
332 double displayMaxRelHumidity(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection)
333 {
334     static const double THRESHOLD = -100;
335     double maxRelHumidity = THRESHOLD;
336     for (int i = 0; i < airQualityCollection.size(); i++){
337         if (airQualityCollection[i].getDate().getYear() == yearInput && airQualityCollection[i].getDate().getMonth() == monthInput) {
338             maxRelHumidity = max(maxRelHumidity, airQualityCollection[i].getRelativeHumidity());
```

```cpp
        }
    }
    return maxRelHumidity;
}

/**
 * Find the highest absolute humidity for that month
 * @param yearInput target year
 * @param monthInput target month
 * @param airQualityCollection vector stores AirQuality objects
 * @return highest absolute humidity for that month
 */
double displayMaxAbsHumidity(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection)
{
    static const double THRESHOLD = -100;
    double maxAbsHumidity = THRESHOLD;
    for (int i = 0; i < airQualityCollection.size(); i++){
        if (airQualityCollection[i].getDate().getYear() == yearInput && airQualityCollection[i].getDate().getMonth() == monthInput) {
            maxAbsHumidity = max(maxAbsHumidity, airQualityCollection[i].getAbsHumidity());
        }
    }
    return maxAbsHumidity;
}

/**
 * Find AirQuality objects with temperature higher than the average temperature for that month
 * @param yearInput target year
 * @param monthInput target month
 * @param airQualityCollection vector stores AirQuality objects
 * @param tempHigherThanAvg vector stores AirQuality objects with temperature higher than the average temperature for that month
 */
void findTempHigherThanAvg(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection, vector<AirQuality>& tempHigherThanAvg)
{
    double avgTemp = calculateAvgTemp(yearInput, monthInput, airQualityCollection);
    // if avg = 0 -> cannot find data corresponding to monthInput
    if (avgTemp == 0)
    {
        return;
    }
    for (int i = 0; i < airQualityCollection.size(); i++){
        if (airQualityCollection[i].getDate().getYear() == yearInput && airQualityCollection[i].getDate().getMonth() == monthInput){
            if (airQualityCollection[i].getTemp() > avgTemp){
                tempHigherThanAvg.push_back(airQualityCollection[i]);
```

```
382             }
383         }
384     }
385 }
386 
387 /**
388  * Find AirQuality objects with relative humidity higher than the average relative humidity for that month
389  * @param yearInput target year
390  * @param monthInput target month
391  * @param airQualityCollection vector stores AirQuality objects
392  * @param relHumidHigherThanAvg vector stores AirQuality objects with relative humidity higher than the average relative humidity for that
    month
393  */
394 void findRelHumidHigherThanAvg(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection, vector<AirQuality>&
    relHumidHigherThanAvg)
395 {
396     double avgRelHumid = calculateRelHumidity(yearInput, monthInput, airQualityCollection);
397     // if avg = 0 -> cannot find data corresponding to monthInput
398     if (avgRelHumid == 0)
399     {
400         return;
401     }
402     for (int i = 0; i < airQualityCollection.size(); i++){
403         if (airQualityCollection[i].getDate().getYear() == yearInput && airQualityCollection[i].getDate().getMonth() == monthInput){
404             if (airQualityCollection[i].getRelativeHumidity() > avgRelHumid){
405                 relHumidHigherThanAvg.push_back(airQualityCollection[i]);
406             }
407         }
408     }
409 }
410 
411 /**
412  * Find AirQuality objects with absolute humidity higher than the average absolute humidity for that month
413  * @param yearInput target year
414  * @param monthInput target month
415  * @param airQualityCollection vector stores AirQuality objects
416  * @param absHumidHigherThanAvg vector stores AirQuality objects with absolute humidity higher than the average absolute humidity for that
    month
417  */
418 void findAbsHumidHigherThanAvg(int yearInput, int monthInput, vector<AirQuality>& airQualityCollection, vector<AirQuality>&
    absHumidHigherThanAvg)
419 {
420     double avgAbsHumid = calculateAbsHumidity(yearInput, monthInput, airQualityCollection);
```

```
421     // if avg = 0 -> cannot find data corresponding to monthInput
422     if (avgAbsHumid == 0)
423     {
424         return;
425     }
426     for (int i = 0; i < airQualityCollection.size(); i++){
427         if (airQualityCollection[i].getDate().getYear() == yearInput && airQualityCollection[i].getDate().getMonth() == monthInput){
428             if (airQualityCollection[i].getAbsHumidity() > avgAbsHumid){
429                 absHumidHigherThanAvg.push_back(airQualityCollection[i]);
430             }
431         }
432     }
433 }
434 
435 /**
436  * Show welcome message to the users
437  */
438 void showWelcomeMsg()
439 {
440     cout << endl;
441     cout << "Welcome to the Air Quality Processor!" << endl;
442     cout << endl;
443 }
444 
445 /**
446  * Show menu to the users
447  */
448 void showMenu()
449 {
450     cout << "*********************************************************************************" << endl;
451     cout << "*                                  MENU                                         *" << endl;
452     cout << "* 0.  Display the average temperature for a certain month.                      *" << endl;
453     cout << "* 1.  Display the average relative humidity for a certain month.                *" << endl;
454     cout << "* 2.  Display the average absolute humidity temperature for a certain month. *" << endl;
455     cout << "* 3.  Display the temperature and temperature at a certain date and time.    *" << endl;
456     cout << "* 4.  Display the highest temperature for a certain month.                   *" << endl;
457     cout << "* 5.  Display the highest relative humidity value for a certain month.       *" << endl;
458     cout << "* 6.  Display the highest absolute humidity for a certain month.             *" << endl;
459     cout << "* 7.  Display temperature higher than the average for a certain month.       *" << endl;
460     cout << "* 8.  Display relative humidity higher than the average for a certain month. *" << endl;
461     cout << "* 9. Display absolute humidity higher than the average for a certain month. *" << endl;
462     cout << "*                                                                               *" << endl;
463     cout << "*********************************************************************************" << endl;
```

```cpp
}

/**
 * valid user's choice, if the user input a letter instead an integer,
 * or the user input a number greater than 10 or less than -1,
 * will print out the error message and let the user re-enter the value
 * @return a valid choice
 */
int userChoice()
{
    cout << endl;
    cout << "--------------------------------------------------------------------------------" << endl;
    cout << "Please select your option by enter the No." << endl
         << "Enter -1 when you finished." << endl;
    int MAX_OPTION = 10;
    bool selecting = true;
    while (selecting)
    {
        int userInput;
        cin >> userInput;
        while (cin.fail())
        {
            cin.clear();
            int const IGNORE_SIZE = 9999999;
            cin.ignore(IGNORE_SIZE, '\n');
            cout << "ERROR: " << endl;
            cout << "Invalid option. Only number accepted. Please Re-enter." << endl;
            showMenu();
            cin >> userInput;
        }
        if (userInput >= -1 && userInput <= MAX_OPTION)
        {
            selecting = false;
            return userInput;
        }
        else
        {
            cout << "ERROR: " << endl;
            cout << "Invalid option. Option out of range. Please Re-enter." << endl;
            showMenu();
        }
    }
    return -1;
```

```
507 }
508
509 /**
510  * Validation check for the month input
511  * The month should be in range of [1, 12], inclusive
512  * @param monthInput - the month need to be checked
513  * @return true if the input value is a valid month
514  */
515 bool validMonth(int monthInput)
516 {
517     int MIN_MONTH = 1;
518     int MAX_MONTH = 12;
519     return (monthInput >= MIN_MONTH && monthInput <= MAX_MONTH);
520 }
521
522
523 /**
524  * validation check for the input time
525  * the valid hour should be in range of [0,24], inclusive
526  * the valid minutes and second should both
527  * be in range of [0, 59], inclusive
528  * @param h - the input hour
529  * @param m - the input minutes
530  * @param s - the input seconds
531  * @return true if the input is a valid time
532  */
533 bool validTime(int h, int m, int s)
534 {
535     int MIN_TIME = 0;
536     int MAX_HOUR = 24;
537     int MAX_MIN_SEC = 59;
538     bool hourValidity = ((h >= MIN_TIME) && (h <= MAX_HOUR));
539     bool minValidity = ((m >= MIN_TIME) && (m <= MAX_MIN_SEC));
540     bool secValidity = ((s >= MIN_TIME ) && (s <= MAX_MIN_SEC));
541     return (hourValidity && minValidity && secValidity);
542 }
543
544 /**
545  * Get a time from the user. will ask the user to enter
546  * the value of hour, minutes, and seconds separately.
547  * Will call validTime function to check whether the input is valid
548  * If there is an letter or the number is out of the range, will
549  * ask the user to input again.
```

```
550  * @return a valid time
551  */
552 Time getValidTime()
553 {
554     bool invalid = true;
555     while (invalid)
556     {
557         int inputHour, inputMin, inputSec;
558         cout << "Please enter the value of hour." << endl;
559         cin >> inputHour;
560         while (cin.fail())
561         {
562             cin.clear();
563             int const IGNORE_SIZE = 9999999;
564             cin.ignore(IGNORE_SIZE, '\n');
565             cout << "ERROR: " << endl;
566             cout << "Invalid option. Only number accepted. Please Re-enter." << endl;
567             cin >> inputHour;
568         }
569         cout << "Please enter the value of minutes." << endl;
570         cin >> inputMin;
571         while (cin.fail())
572         {
573             cin.clear();
574             int const IGNORE_SIZE = 9999999;
575             cin.ignore(IGNORE_SIZE, '\n');
576             cout << "ERROR: " << endl;
577             cout << "Invalid option. Only number accepted. Please Re-enter." << endl;
578             cin >> inputMin;
579         }
580         cout << "Please enter the value of seconds." << endl;
581         cin >> inputSec;
582         while (cin.fail())
583         {
584             cin.clear();
585             int const IGNORE_SIZE = 9999999;
586             cin.ignore(IGNORE_SIZE, '\n');
587             cout << "ERROR: " << endl;
588             cout << "Invalid option. Only number accepted. Please Re-enter." << endl;
589             cin >> inputSec;
590         }
591         if (validTime(inputHour, inputMin, inputSec))
592         {
```

```
593             invalid = false;
594             return Time(inputSec, inputMin, inputHour);
595         }
596         else
597         {
598             cout << "Invalid combination, please Re-enter." << endl;
599         }
600     }
601     return Time();
602 }
603 
604 /**
605  * Get a valid year from the user
606  * @return a valid year
607  */
608 int getValidInputYear()
609 {
610     int yearInput;
611     cout << "Please enter a year: " << endl;
612     cin >> yearInput;
613     while (cin.fail())
614     {
615         cin.clear();
616         int const IGNORE_SIZE = 9999999;
617         cin.ignore(IGNORE_SIZE, '\n');
618         cout << "ERROR: " << endl;
619         cout << "Invalid option. Only number accepted. Please Re-enter." << endl;
620         cin >> yearInput;
621     }
622     return yearInput;
623 }
624 
625 /**
626  * get a valid month from the user
627  * @return a valid month
628  */
629 int getValidInputMonth()
630 {
631     int monthInput;
632     cout << "Please enter a month: " << endl;
633     cin >> monthInput;
634     while (cin.fail())
635     {
```

```
636         cin.clear();
637         int const IGNORE_SIZE = 9999999;
638         cin.ignore(IGNORE_SIZE, '\n');
639         cout << "ERROR: " << endl;
640         cout << "Invalid option. Only number accepted. Please Re-enter." << endl;
641         cin >> monthInput;
642     }
643     while (!validMonth(monthInput)) {
644         cout << "Month should be in range of [1, 12], inclusive" << endl;
645         cout << "Please re-enter." << endl;
646         cin >> monthInput;
647         while (cin.fail()) {
648             cin.clear();
649             int const IGNORE_SIZE = 9999999;
650             cin.ignore(IGNORE_SIZE, '\n');
651             cout << "ERROR: " << endl;
652             cout << "Invalid option. Only number accepted. Please Re-enter." << endl;
653             cin >> monthInput;
654         }
655     }
656     return monthInput;
657 }
658
659 /**
660  * validation check for the day of the date
661  * will take into consideration of different day in each month and the
662  * leap year.
663  * @param inputYear - the year user enter
664  * @param inputMonth - the month user enter
665  * @param inputDay - the day user enter
666  * @return - true if the day is in a valid day for the certain month and year
667  */
668 bool validDay(int inputYear, int inputMonth, int inputDay) {
669     int validDay[] = {31,28,31,30,31,30,31,31,30,31,30,31};
670     if ((inputYear % 4 == 0) && (inputYear % 100 != 0) || (inputYear % 400 == 0))
671     {
672         int februaryIndex = 2;
673         validDay[februaryIndex] = 29;
674     }
675     int SHIFT = 1;
676     return ((inputDay > 0) && (inputDay <= (validDay[(inputMonth) - SHIFT])));
677 }
678
```

```cpp
/**
 * get a valid day from the user
 * will call the function validDAy to check whether there is a certion day
 * in a certain month
 * @return a valid day
 */
int getValidInputDay(int inputYear, int inputMonth) {
    int dayInput;
    cout << "Please enter a day: " << endl;
    cin >> dayInput;
    while (cin.fail())
    {
        cin.clear();
        int const IGNORE_SIZE = 9999999;
        cin.ignore(IGNORE_SIZE, '\n');
        cout << "ERROR: " << endl;
        cout << "Invalid option. Only number accepted. Please Re-enter." << endl;
        cin >> dayInput;
    }
    while (!validDay(inputYear,inputMonth,dayInput)) {
        cout << "Input day is not valid, there is no " << dayInput << " in the month of " << inputMonth << endl;
        cout << "Please re-enter." << endl;
        cin >> dayInput;
        while (cin.fail()) {
            cin.clear();
            int const IGNORE_SIZE = 9999999;
            cin.ignore(IGNORE_SIZE, '\n');
            cout << "ERROR: " << endl;
            cout << "Invalid option. Only number accepted. Please Re-enter." << endl;
            cin >> dayInput;
        }
    }
    return dayInput;
}

/**
 * call different function based on the user's choice
 * @param option - the valid user's choice
 * @param airQualityCollection - the dataSet need to be handled
 */
void doUserChoice(int option, vector<AirQuality>& airQualityCollection)
{
    int EXIT = -1;
```

```
722     if (option == EXIT)
723     {
724         return;
725         // exit from the function
726     }
727     int yearInput = getValidInputYear();
728     int monthInput = getValidInputMonth();
729     if (option == 3)
730     {
731         int dayInput = getValidInputDay(yearInput, monthInput);
732         Date curDay = Date(dayInput, monthInput, yearInput);
733         Time curTime = getValidTime();
734         double curTemp, curRelHumidity;
735         doFindTempAndHum(curDay, curTime, curTemp, curRelHumidity, airQualityCollection);
736     }
737     else
738     {
739         if (option == 0 || option == 1 || option == 2 ||
740             option == 4 || option == 5 || option == 6)
741         {
742 
743             doCalOrDis(option, yearInput, monthInput, airQualityCollection);
744         }
745         else
746         {
747             vector<AirQuality> res;
748             doCalThanAvg(option, yearInput, monthInput, airQualityCollection, res);
749         }
750     }
751 }
752 
753 /**
754  * Calculation or display. Will be called if the user's choice is
755  * 0,1,2,4,5,or 6.
756  * @param option - the choice
757  * @param yearInput - year
758  * @param monthInput - month
759  * @param airQualityCollection - the data set
760  */
761 void doCalOrDis(int option, int yearInput, int monthInput,vector<AirQuality>& airQualityCollection)
762 {
763     double res;
764     int NO_RECORD_CAL = 0;
```

```
765     int NO_RECORD_DIS = -100;
766     switch (option) {
767         case 0:
768             res = calculateAvgTemp(yearInput, monthInput, airQualityCollection);
769             if (res == NO_RECORD_CAL)
770             {
771                 cout << "No records found." << endl;
772             }
773             else
774             {
775                 cout << "The average temperature is: " << res << endl;
776             }
777             break;
778         case 1:
779             res = calculateRelHumidity(yearInput, monthInput, airQualityCollection);
780             if (res == NO_RECORD_CAL)
781             {
782                 cout << "No records found." << endl;
783             }
784             else
785             {
786                 cout << "The average relative humidity is: " << res << endl;
787             }
788             break;
789         case 2:
790             res = calculateAbsHumidity(yearInput, monthInput, airQualityCollection);
791             if (res == NO_RECORD_CAL)
792             {
793                 cout << "No records found." << endl;
794             }
795             else
796             {
797                 cout << "The average absolute humidity is: " << res << endl;
798             }
799             break;
800         case 4:
801             res = displayMaxTemp(yearInput, monthInput, airQualityCollection);
802             if (res == NO_RECORD_DIS)
803             {
804                 cout << "No records found." << endl;
805             }
806             else
807             {
```

```
808                 cout << "The highest temperature is: " << res << endl;
809             }
810             break;
811         case 5:
812             res = displayMaxRelHumidity(yearInput, monthInput, airQualityCollection);
813             if (res == NO_RECORD_DIS)
814             {
815                 cout << "No records found." << endl;
816             }
817             else
818             {
819                 cout << "The highest relative humidity is: " << res << endl;
820             }
821             break;
822         case 6:
823             res = displayMaxAbsHumidity(yearInput, monthInput, airQualityCollection);
824             if (res == NO_RECORD_DIS)
825             {
826                 cout << "No records found." << endl;
827             }
828             else
829             {
830                 cout << "The highest absolute humidity is: " << res << endl;
831             }
832             break;
833     }
834 }
835 
836 /**
837  * calculation certain data greater than average
838  * @param option - user's choice
839  * @param yearInput - year
840  * @param monthInput - month
841  * @param airQualityCollection - dataSet
842  * @param res - temporary container
843  */
844 void doCalThanAvg(int option, int yearInput, int monthInput, vector<AirQuality>& airQualityCollection, vector<AirQuality>& res)
845 {
846     switch (option) {
847         case 7:
848             findTempHigherThanAvg(yearInput, monthInput, airQualityCollection, res);
849             if (res.size() == 0)
850             {
```

```
851                 cout << "No records found." << endl;
852             }
853             else
854             {
855                 cout << "The dates and times when temperature is higher than the average are listed below: " << endl;
856                 for (int i = 0; i < res.size(); i++)
857                 {
858                     cout << "Date: " << setw(2) << res[i].getDate() << setw(10)
859                          << "Time: " << setw(2) << res[i].getTime() << setw(18)
860                          << "Temperature: " << res[i].getTemp() << endl;
861                 }
862             }
863             break;
864         case 8:
865             findRelHumidHigherThanAvg(yearInput, monthInput, airQualityCollection, res);
866             if (res.size() == 0)
867             {
868                 cout << "No records found." << endl;
869             }
870             else
871             {
872                 cout << "The dates and times when relative humidity is higher than the average are listed below: " << endl;
873                 for (int i = 0; i < res.size(); i++)
874                 {
875                     cout << "Date: " << setw(2) << res[i].getDate() << setw(10)
876                          << "Time: " << setw(2) << res[i].getTime() << setw(24)
877                          << "Relative Humidity: " << res[i].getRelativeHumidity() << endl;
878                 }
879             }
880             break;
881         case 9:
882             findAbsHumidHigherThanAvg(yearInput, monthInput, airQualityCollection, res);
883             if (res.size() == 0)
884             {
885                 cout << "No records found." << endl;
886             }
887             else
888             {
889                 cout << "The dates and times when absolute humidity is higher than the average are listed below: " << endl;
890                 for (int i = 0; i < res.size(); i++)
891                 {
892                     cout << "Date: " << setw(2) << res[i].getDate() << setw(10)
893                          << "Time: " << setw(2) << res[i].getTime() << setw(24)
```

```
894                             << "Absolute Humidity: " << res[i].getAbsHumidity() << endl;
895                 }
896             }
897             break;
898     }
899 }
900
901 /**
902  * Find a exertion data.
903  * @param inputDate - input date
904  * @param inputTime - input time
905  * @param currentTemp - current temp
906  * @param currentRelHumidity - current relative humidity
907  * @param airQualityCollection - dataSet
908  */
909 void doFindTempAndHum(Date inputDate, Time inputTime, double &currentTemp, double &currentRelHumidity, vector<AirQuality>&
    airQualityCollection)
910 {
911     displayTempAndRelHumidity(inputDate, inputTime, currentTemp, currentRelHumidity,airQualityCollection);
912     if (currentTemp == 0 && currentRelHumidity == 0)
913     {
914         cout << "No records found." << endl;
915     }
916     else
917     {
918         cout << "At that date and time, temp is: " << currentTemp << ", rel humid is: " << currentRelHumidity << endl;
919     }
920 }
921
922
```

File - C:\Users\nicho\Desktop\GitHub\CS5008_LastAssignment\AirQualityProcessor\final_project\Time.cpp

```cpp
#include <iostream>
#include "Time.h"

using namespace std;

// default constructor
Time::Time()
{
    sec = 1;
    min = 1;
    hour = 1;
}

// specific constructor
Time::Time(int s, int m, int h)
{
    sec = s;
    min = m;
    hour = h;
}

// destructor
Time::~Time() {

}

// setters
void Time::setSec(int s)
{
    sec = s;
}

void Time::setMin(int m)
{
    min = m;
}

void Time::setHour(int h)
{
    hour = h;
}

void Time::setSmh(int s, int m, int h)
```

```
44 {
45     sec = s;
46     min = m;
47     hour = h;
48 }
49
50 // getters
51 int Time::getSec() const
52 {
53     return sec;
54 }
55
56 int Time::getMin() const
57 {
58     return min;
59 }
60
61 int Time::getHour() const
62 {
63     return hour;
64 }
65
66 // printf
67 void Time::print() const
68 {
69     cout << sec << "/"
70          << min << "/"
71          << hour;
72 }
73
74 // ostream operator
75 ostream& operator << (ostream& osObject, const Time& time1) //creating an instance of a date class, i.e. date1
76 {
77     if (time1.min == 0 && time1.sec == 0){
78         osObject << time1.hour
79                  << ":" << "00"
80                  << ":" << "00";
81     }
82     else {
83         osObject << time1.hour
84                  << ":" << time1.min
85                  << ":" << time1.sec;
86     }
```

```
 87     return osObject;
 88 }
 89
 90 // istream operator
 91 istream& operator >> (istream& isObject, Time& time1)
 92 {
 93     isObject >> time1.hour >> time1.min >> time1.sec;
 94     return isObject;
 95 }
 96
 97 // Overloading "equal to" operator
 98 bool Time::operator==(const Time& otherTime) const
 99 {
100     if (sec == otherTime.sec && min == otherTime.min
101         && hour == otherTime.hour)
102         return true;
103     else
104         return false;
105 }
106
107 // Overloading "not equal to" operator
108 bool Time::operator!=(const Time& otherTime) const
109 {
110     return !(*this == otherTime);
111 }
112
113 // Overloading "less than or equal to" operator
114 bool Time::operator<=(const Time& otherTime) const
115 {
116     return (*this < otherTime || *this == otherTime);
117 }
118
119 // Overloading "less than" operator
120 bool Time::operator<(const Time& otherTime) const
121 {
122     if ((hour < otherTime.hour) ||
123         (hour == otherTime.hour && min < otherTime.min) ||
124         (hour == otherTime.hour && min == otherTime.min && sec < otherTime.sec))
125         return true;
126     else
127         return false;
128 }
129
```

```
130 // Overloading "greater than or equal to" operator
131 bool Time::operator>=(const Time& otherTime) const
132 {
133     return !(*this < otherTime);
134 }
135
136 // Overloading "greater than" operator
137 bool Time::operator>(const Time& otherTime) const
138 {
139     return !(*this <= otherTime);
140 }
```

File - C:\Users\nicho\Desktop\GitHub\CS5008_LastAssignment\AirQualityProcessor\final_project\AirQuality.h

```
1  //
2  // Created by hycwy on 4/10/2022.
3  //
4  #include "Date.h"
5  #include "Time.h"
6
7
8  #ifndef FINAL_PROJECT_AIRQUALITY_H
9  #define FINAL_PROJECT_AIRQUALITY_H
10
11 class AirQuality {
12     friend ostream& operator << (ostream&, const AirQuality&);
13     friend istream& operator >> (istream&, AirQuality&);
14
15 public:
16     AirQuality();
17
18     AirQuality(Date dateInput, Time timeInput, double t, double r, double a);
19
20     AirQuality(double t, double r, double a);
21
22     ~AirQuality();
23
24     void setDate(Date d);
25     void setTime(Time t);
26     void setTemp(double t);
27     void setRelativeHumidity(double r);
28     void setAbsHumidity(double a);
29
30     Date getDate();
31     Time getTime();
32     double getTemp();
33     double getRelativeHumidity();
34     double getAbsHumidity();
35
36 private:
37
38     // data members
39     Date date = Date();
40     Time time = Time();
41     double temp;
42     double relativeHumidity;
43     double absHumidity;
```

```
44 };
45
46
47 #endif //FINAL_PROJECT_AIRQUALITY_H
48
```

File - C:\Users\nicho\Desktop\GitHub\CS5008_LastAssignment\AirQualityProcessor\final_project\AirQuality.cpp

```cpp
//
// Created by hycwy on 4/10/2022.
//

#include "AirQuality.h"
#include "Date.h"
#include "Time.h"

AirQuality::AirQuality() {
    temp = 0.0;
    relativeHumidity = 0.0;
    absHumidity = 0.0;
}

AirQuality::AirQuality(double t, double r, double a) {
    temp = t;
    relativeHumidity = r;
    absHumidity = a;
}

AirQuality::AirQuality(Date dateInput, Time timeInput, double t, double r, double a) {
    date.setDmy(dateInput.getDay(),dateInput.getMonth(),dateInput.getYear());
    time.setSmh(timeInput.getSec(), timeInput.getMin(), timeInput.getHour());
    temp = t;
    relativeHumidity = r;
    absHumidity = a;
}

AirQuality::~AirQuality() {}

void AirQuality::setDate(Date d) {
    date.setDmy(d.getDay(), d.getMonth(), d.getYear());
}

void AirQuality::setTime(Time t) {
    time.setSmh(t.getSec(), t.getMin(), t.getHour());
}

void AirQuality::setTemp(double t) {
    temp = t;
}

void AirQuality::setRelativeHumidity(double r) {
```

File - C:\Users\nicho\Desktop\GitHub\CS5008_LastAssignment\AirQualityProcessor\final_project\AirQuality.cpp

```
44      relativeHumidity = r;
45  }
46
47  void AirQuality::setAbsHumidity(double a) {
48      absHumidity = a;
49  }
50
51  Date AirQuality::getDate() {
52      return date;
53  }
54
55  Time AirQuality::getTime() {
56      return time;
57  }
58
59  double AirQuality::getAbsHumidity() {
60      return absHumidity;
61  }
62
63  double AirQuality::getRelativeHumidity() {
64      return relativeHumidity;
65  }
66
67  double AirQuality::getTemp() {
68      return temp;
69  }
70
71  ostream& operator << (ostream& osObject, const AirQuality& Air1)
72  {
73      osObject << Air1.date << " " << Air1.time << " " << Air1.temp << " " << Air1.relativeHumidity << " " << Air1.absHumidity;
74      return osObject;
75  }
76
77
78  istream& operator >> (istream& isObject, AirQuality& Air1)
79  {
80      isObject >> Air1.date >> Air1.time >> Air1.temp >> Air1.relativeHumidity >> Air1.absHumidity;
81      return isObject;
82  }
```

File - C:\Users\nicho\Desktop\GitHub\CS5008_LastAssignment\AirQualityProcessor\final_project\CMakeLists.txt

```
cmake_minimum_required(VERSION 3.21)
project(final_project)

set(CMAKE_CXX_STANDARD 14)

add_executable(final_project main.cpp)

```