CSci 1113: Introduction to C/C++ Programming for Scientists and Engineers Homework 7

Due Date: Tuesday, Apr. 9 before 5:30pm.

Purpose: This homework involves practice with classes and objects.

Instructions: This is an *individual* homework assignment. *Please follow the rules for individual work as explained in the recent class presentations and the academic conduct document and quiz.* There are *two* problems worth 30 points each. The rules and reminders are the same as for past homework, so see the previous homework write-ups if you need a reminder.

Advice: While the code for this assignment is not particularly complicated, there is a good amount of it. So start early, ask for help during office hours as needed, and develop your code incrementally.

Problem A: How's the Weather at Elysium Planitia?[1] (30 points)

A large amount of scientific data is gathered by small sensors, including sensors studying ocean conditions, sensors studying weather over land, and sensors on space probes. This problem asks you to implement a class for a sensor *reading*. Specifically, write a class `SensorReading` with the following member variables:

- `time`: an int giving the time the reading was taken, measured in seconds since midnight.
- `pressure`: a double giving the pressure in Pascals (Pa).
- `windDirection`: a double giving the wind direction as an angle between 0.0 degrees (inclusive) and 360.0 degrees (exclusive). You can assume 0.0 degrees corresponds to north, and the angles proceed counterclockwise, so due west should be 90.0 degrees, for example. Also assume the reading is in degrees and fractions of degrees (i.e., minutes are not used here as parts of degrees), so, e.g., 36.5 is 36 and a half degree.
- `windSpeed`: a double giving the speed of the wind in meters per second.

The class should have the following member functions:

- A default constructor that sets all the member variables to 0.
- A constructor `sensorReading(int t, double pr, double wd, double ws)` that initializes the member variables to the corresponding arguments.
- `void getTime(int& hours, int& minutes, int& seconds) const;` computes the time in 24 hour format and stores it in the call-by-reference parameters. That is, the number of hours will be between 0 and 23, inclusive, the number of minutes between 0 and 59, inclusive, and the number of seconds between 0 and 59, inclusive. For example, if the time is 7802, then `hours` should be 2, minutes should be 10, and seconds should be 2. You may assume this function will only be called when the time is in the valid range (i.e., it yields a valid 24 hour time).

[1]See `https://mars.nasa.gov/insight/weather/`. However, the problem here isn't directly patterned after the Elysium Planitia weather readings, but only after various weather sensor readings in general.

- `bool validate() const;` checks the following conditions are met: (i) the time in seconds is nonnegative and is strictly less than the number of seconds in a day, namely $60 \times 60 \times 24$. (ii) The pressure is nonnegative. (iii) The wind direction is at least 0.0 and strictly less than 360.0. (iv) The wind speed is nonnegative. If all these conditions are met the function should return true; otherwise it should return false.

- `void print( ) const;` prints all the information for the sensor reading as shown in the examples below. Note if any reading is out of the valid range (see the ranges in the `validate` function immediately above), then this `print` function should print "Out of range" for that variable.

- `void adjustWindDirection(double change);` Adjusts the wind direction by adding `change` degrees. Do not make any assumptions about the value of `change`. For example, `change` might be negative — this is fine, just add the negative amount. Also note adding `change` might create a direction outside the valid range; in this case adjust the direction further so it is the equivalent angle but in the range [0.0, 360.0). For example, if the direction is originally 355.0 and the change is 10.4 degrees, then the result should be 5.4 degrees, not 365.4.

- `void adjustTime(int change);` Adjusts the time by adding `change` seconds. Do not make any assumptions about the value of `change` (other than it is an int). For example, `change` might be negative; this is fine, just add the negative amount. As an example, if `time` is 5280 and `change` is -10, then `time` should become 5270. Also note the resulting time might be outside the valid range; however, in this case this function should still adjust the time, but *not* adjust it further to put it into the valid range. Example: if `time` is 20 and `change` is -30 then `time` should become -10.

Then write a main program that does the following:

1. Sets up a `sensorReading` object, with time 6722, pressure 724.5, wind direction 340.8, and wind speed 5.2.

2. Asks the user to input any of the following options:

    - `v`: Validate the sensor reading. Print "Valid" if the reading is valid, and print "NOT valid" otherwise.
    - `p`: Print the sensor reading data in the format shown in the example below.
    - `t`: Adjust the time by asking the user to input the time change amount (in seconds) and then adding that amount to the sensor reading time.
    - `d`: Adjust the wind direction by asking the user to input the direction change amount and then adding that amount to the sensor reading wind direction.
    - `q`: Quit.
    - If the user inputs any other character, the program should print "That character is not an option".

    Continue this step repeatedly until the user indicates they wish to quit.

Here is an example:

```
Validate(v), print(p), adjust time(t), adjust direction(d), or quit(q):  p
Time:  1:52:02
Pressure:  724.5 Pa
Direction:  340.8 Degrees
Speed:  5.2 m/s
Validate(v), print(p), adjust time(t), adjust direction(d), or quit(q):  v
Valid
Validate(v), print(p), adjust time(t), adjust direction(d), or quit(q):  y
That character is not an option
Validate(v), print(p), adjust time(t), adjust direction(d), or quit(q):  d
Amount of change:  30.4
Validate(v), print(p), adjust time(t), adjust direction(d), or quit(q):  t
Amount of change:  -8000
Validate(v), print(p), adjust time(t), adjust direction(d), or quit(q):  p
Out of range
Pressure:  724.5 Pa
Direction:  11.2 Degrees
Speed:  5.2 m/s
Validate(v), print(p), adjust time(t), adjust direction(d), or quit(q):  v
NOT valid
Validate(v), print(p), adjust time(t), adjust direction(d), or quit(q):  q
```

Additional clarification, requirements, and advice:

- Note the `print()` member function does not need to do any special formatting beyond using the labels above, putting each item on a different line, and printing the time in the format H:MM:SS (if the number of hours is between 0 and 9) or HH:MM:SS if the number of hours is between 10 and 23. Note, however, that this might require adding a leading zero for the number of minutes or seconds if their value is between 0 and 9.
- The tasks here should not be difficult, but there is a fair amount of code to write. Implement your class incrementally: add member functions a few at a time, test them, and get them working before implementing additional member functions.
- Problem B below asks you to add some member functions to your class. However, in doing this problem, Problem A, only use the member functions above; do not use any you add in Problem B.

See the submission details at the end of this section about how to submit your solution.

Problem B: Many Readings (30 points)

In this problem you will add to, and use further, the `SensorReading` class you developed in Problem A. To begin add the following member functions to the class:

- `void set(int t, double pr, double wd, double ws)`: Sets the member variables `time`, `pressure`, `windDirection`, and `windSpeed` to `t`, `pr`, `wd` and `ws`, respectively.
- `void printCSV(ostream& out) const;` This will print the class data information in the following format:

```
time,pressure,windDirection,windSpeed
```

It should just print these values as they are, with commas separating them. It should *not* do any additional formatting such as adding spaces, formatting the numbers using any commands to set the field width, etc.

Then write a main program that does the following:

1. Sets up an array of 10 `SensorReading` objects using the default constructor.
2. Asks a user for an input filename and tries to open the file. If the file opening is unsuccessful the program should print "Unable to open file" and terminate using `exit(1)`.
3. Reads the data in the file and updates the sensor reading objects using the `set` member function. Each line in the file has four numbers: the time (in seconds), pressure, wind direction, and wind speed. All are separated by a single space. You can assume the file has this format, and has at least 1 line of data, and no more than 10 lines of data.
4. Closes the input file.
5. Reads in an output file name, and tries to open the file. If the file opening is unsuccessful the program should print "Unable to open file" and terminate using `exit(1)`.
6. Goes through the array of sensors for which there was a reading and does the following:
   - Subtracts 24 seconds from each reading (the clock got started early).
   - Checks if each reading (after the time adjustment) is valid.
   - Writes the information for any *valid* reading to the output file using the `printCSV` member function.
7. Prints to the terminal (i) the total number of invalid readings from the previous step, and (iii) the total number of readings that were written to the output file in the previous step. (See the example below.)
8. Closes the output file.

Additional comments, clarifications, and hints:

- The loop in the program should only go through the sensors for which there was a reading. For example, if there were 7 lines in the input data file, then there are 7 readings, and your program should analyze those 7, not 10.
- Note that any invalid reading should not be written out to the file.
- It is possible that there are no valid readings in the file; in this case the output file should still be opened (and later closed) but nothing will be written to it so it will be empty.
- Note some output goes to the output file, and some to the terminal.
- As usual, develop your code incrementally

Here is an example. First, here is the input file:

```
7612 720.0 23.9 4.85
7889 721.1 24.7 4.9
7900 720.2 24.2 -3.5
7910 720.2 24.1 4.8
8909898 720.2 24.2 4.9
17920 720.7 25.2 5.0
```

Here is the terminal input/output. As usual, user input is underlined, and there is a single space between any prompt or label and the resulting input or output.

```
Input file:  testIn.txt
Output file:  testOut.txt
Number of invalid readings:  2
Number of readings written to output file:  4
```

And here is the output file:

```
7588,720,23.9,4.85
7865,721.1,24.7,4.9
7886,720.2,24.1,4.8
17896,720.7,25.2,5
```

Submitting Your Solution (IMPORTANT!)

Since this homework involves a number of files, some of which are used for both Problems A and B, the submission process will be slightly different. So read this part carefully and ask if you have any questions on it or have trouble with this submission process.

You solutions will involve four files:

- **<YourName>_7A.cpp**: Your Problem A main file. Here, as usual, replace <YourName> with your U of M email address.
- **<YourName>_7B.cpp**: Your Problem B main file. Here, as usual, replace <YourName> with your U of M email address.
- **SensorReading.hpp**: Your class declaration file. Note you should not put your U of M email address in this file name.
- **SensorReading.cpp**: Your class implementation file. Note you should not put your U of M email address in this file name.

(Note you should only submit one copy of the class .hpp and .cpp files — you do *not* submit these files twice.)

To submit these files, zip them together into a single file by typing the following command from a terminal window. Make sure you are in the same directory (folder) as the files for this homework.

```
zip <YourName>.zip  <YourName>_7A.cpp <YourName>_7B.cpp SensorReading.hpp SensorReading.cpp
```

where, as usual, you replace <YourName> with your U of M email address. Then submit the resulting file `<YourName>.zip` using the Homework 7 Submission link. Note that once you zip the four files into a single file, you just need to submit the single zip file.

If you have any trouble with the zip command, please check with any of the TAs during office hours. The zip command should work on any of the lab machines. It might also work on personal laptops, but if it does not, you will either need to transfer all your files to a lab machines, zip them there, and then submit from there, or find out how to zip your files on your laptop.