CSC 111 - FALL 2020
FUNDAMENTALS OF PROGRAMMING WITH ENGINEERING APPLICATIONS
ASSIGNMENT 6
UNIVERSITY OF VICTORIA

Due: Sunday, Dec. 6th, 2020 at 11:59pm. **Late assignments will not be accepted.**

This assignment will be submitted electronically through conneX (as described in 'Submission Instructions' below). All code submissions must be your own work. Sending or receiving code from other students is plagiarism.

1 Overview

This assignment involves performing simple data analysis on large datasets of temperature observations from around Vancouver Island. The main goal of the assignment is to develop skills with structured data and file input/output.

Your submission will be a library implementation (similar to Assignment 4). A header file `temperature_data.h` and a blank `temperature_data.c` file have been provided, along with several tester programs that use various functions from the library.

Some of the functions you write will work with input files containing set of temperature observations. The format for these input files is described in Section 2 below. A set of sample input files has been posted to conneX.

Best Practices for Code Style

Your submissions **must** observe the following guidelines. Submissions which do not meet the guidelines will have marks deducted, even if they function correctly (although some of the guidelines, such as those relating to uninitialized memory, might also affect the ability of the program to work correctly on a reliable basis, which could result in a deduction for both style and function).

- Submitted files must be syntactically correct and named according to the assignment specification.
- Every source file must contain a comment (at or near the top of the file) with your name, student number, the date the code was written and a brief description of the program.
- The `goto` statement is not permitted in any assignment submissions. Instead, your code should use structured control flow elements, like `if`, `for` and `while`.
- Global variables (data variables created outside of the scope of a function) are not permitted, except when explicitly allowed by the assignment. For this assignment, no global variables are permitted, except for `const` variables if needed.
- Every function with a non-`void` return type must return a value.
- Uninitialized variables may not be used before being assigned a value.
- Every file opened (by functions such as `fopen`) **must** be explicitly closed (by a call to `fclose`).

- Any dynamically allocated memory must be freed explicitly. For this assignment, it should not be necessary to use any dynamically allocated memory.

2 Input Format

The Vancouver Island School-Based Weather Station Network (VWSN)[1] maintains a set of weather stations at various public sites around Vancouver Island, primarily in elementary and secondary schools, but also at scientific institutions and universities. Each station is identified by a numerical station identifier. For example, there are three weather stations active on the UVic campus: Station 1 is located on the roof of the Ian Stewart Complex, station 143 is located on the Earth, Ocean and Atmospheric Sciences building and station 163 is located on the David Turpin building.

Each weather station reports observations of current conditions for a variety of scientific variables (temperature, barometric pressure, humidity, wind speed/direction, etc.) to a centralized server at regular intervals, and the collected data is publicly available. For this assignment, we will work with datasets containing temperature observations from all of the active stations in the network (about 150 - 200 at any given time) at a resolution of about five minutes. You should consider that if 150 stations report data every five minutes for an entire day, there will be $150 \cdot \frac{60}{5} = 1800$ observations per hour and $150 \cdot \frac{60}{5} \cdot 24 = 43,200$ observations for the whole day. Some of our input files will involve multiple days of data, so you will be writing programs that can handle hundreds of thousands of data points.

For this assignment, the VWSN temperature data has been converted into text files where each line describes one temperature observation (at a particular time, from a particular station). Each line of the file has the form

```
year    month    day    hour    minute    station_number    temperature
```

For example, the following sample contains several temperature observations in the format above. Except for the temperature values (which are floating point values), all of the values in the file are integers.

```
2020 11 14 12 07 1 8.7
2020 11 14 11 58 143 8.6
2020 11 14 13 03 163 8.9
2020 11 13 18 06 1 7.3
```

The first line contains an observation of 8.7 degrees Celsius from station number 1 at 12:07 on November 14th, 2020. The third line contains an observation of 8.9 degrees Celsius from station number 163 at 13:03 on November 14th, 2020. The hour and minute columns specify a 24-hour time (so 12:05 is five minutes after noon). Notice that the rows are in no particular order (and you should not assume that they will be sorted by date, or by station number, or by temperature).

A data file may contain any number of lines, spanning any number of months and days and any set of stations. You may make the following assumptions.

- In any valid file, every line contains all seven data fields (year, month, day, hour, minute, station number and temperature).

1. `http://www.victoriaweather.ca`

- There is no requirement that every station report observations in every file (and, in fact, some station numbers will never appear since the associated station has been decommissioned), but you may assume that the every station ID number will be greater than 0 and less than or equal to 251.
- Temperatures in degrees Celsius may be positive, zero or negative, so your programs should not assume that all temperatures are positive. Due to the local climate, there are usually relatively few negative temperature observations, but they do occur (and will be part of the testing suite).
- Most of the provided test data files contain observations from late October and early November 2020, but some files contain dates in other months (one file contains a sample of data from an entire year). You may assume that month numbers are always between 1 and 12 (inclusive) and that day numbers are always between 1 and 31 (inclusive). You do not have to verify that a given month contains the right number of days (so if you see an observation from February 31st, which shouldn't exist, you can treat it just like any other observation, although no such observations will be present in the provided files).

You can read numerical data from the input files by opening them with `fopen` and using the `fscanf` function (with appropriate formatting specifiers) to read data. You are **strongly encouraged** to avoid the use of raw input functions like `fgetc` or `fgets` on this assignment (since `fscanf` is the only library function that can reliably read formatted numerical data from a file). You are also advised to avoid using `feof` in your solution (since you can use the return value of `fscanf` to decide whether any input was read).

3 Your Task

A header file `temperature_data.h` and a blank implementation `temperature_data.c` have been posted to conneX. The library header contains `typedef` statements to created structure types for a date (year/month/day) and a single observation (containing a date, time, station number and temperature). The library functions work with these observation objects in various ways (e.g. reading observations from a file or processing arrays of observations). Your task is to implement each function in the library according to its specification.

You may **not** make any assumptions about the structure of input data except those explicitly given in this specification document or in the specification comments in the library header file. Except in the `print_station_extremes` and `print_daily_averages` functions (which are explicitly required to produce output to the user), the library functions must not print any output to the user.

As with Assignment 4, since your submission will be a library implementation, your submission must **not** contain an implementation of `main()`, since that will make it impossible for us to test. Any submissions which contain an implementation of `main()` will receive a mark of zero.

Important Note: Before starting on the `print_station_extremes` and `print_daily_averages` data analysis functions, make sure that all of the other library functions (which read observations from a file) work correctly. Unless the input functions (`count_observations`, `read_observation` and `load_all_observations`) work correctly, it will not be possible for us to test your data analysis functions.

3.1 Tester Programs

Several tester programs have been provided to help you debug and test your implementation.

- `a6_tester_basic.c`: Creates several `Observation` objects directly (without using the File I/O library functions) and tests the `print_station_extremes` and `print_daily_averages` functions.
- `a6_tester_read_observations.c`: Reads `Observation` objects from a file using the `read_observation` function and prints each observation out. This might be helpful for validating your `read_observation` implementation.
- `a6_tester_load_all.c`: Reads `Observation` objects from a file using the `load_all_observations` function and prints each observation out. This might be helpful for validating your `count_observations` and `load_all_observations` implementations.
- `a6_tester_all.c`: Reads `Observation` objects from a file using the `load_all_observations` and then uses `print_station_extremes` and `print_daily_averages` to analyze the complete set of observations.

To run each tester program, you can compile its source file together with your library implementation. For example, to compile with the `a6_tester_basic.c` program, you can use the command

```
gcc -Wall -std=c11 -o a6_tester_basic a6_tester_basic.c temperature_data.c
```

3.2 Daily Averages

Consider the small input sample below.

```
2020 11 14 12 07 1 8.7
2020 11 14 11 58 143 8.6
2020 11 14 13 03 163 8.9
2020 11 13 18 06 1 7.3
2020 11 13 19 03 1 7.3
2020 12 1 01 06 2 -1.0
2020 12 1 09 07 2 -0.5
2020 6 10 17 00 111 18.7
```

Notice that there are observations present for 4 different days (November 14th, November 13th, December 1st and June 10th). The purpose of the `print_daily_averages` function is to take an array of `Observation` objects and print out a summary of the average temperature for each distinct date (year/month/day) in the dataset.

The output will have one line for each date in the input file. Each line will have the format "`year month day average_temperature`", where `temperature` is printed with one decimal place (using the '`%.1f`' specifier). The output must be printed **ordered by date** (with earlier dates appearing first). Dates for which no observations were recorded must be omitted from the output. Dates for which there are at least one observation must appear exactly once in the output (printing multiple lines of output for the same date is incorrect).

For example, on the set of observations taken from the input file shown above, the `print_daily_averages` function would print the following output (and **nothing else**).

```
2020 6 10 18.7
2020 11 13 7.3
```

```
2020 11 14 8.7
2020 12 1 -0.8
```

3.3 Station Extremes

Again consider the input sample below from Section 3.2.

```
2020 11 14 12 07 1 8.7
2020 11 14 11 58 143 8.6
2020 11 14 13 03 163 8.9
2020 11 13 18 06 1 7.3
2020 11 13 19 03 1 7.3
2020 12 1 01 06 2 -1.0
2020 12 1 09 07 2 -0.5
2020 6 10 17 00 111 18.7
```

The data contains records for 5 station IDs (1, 2, 111, 143 and 163). Some stations only have one reported observation and some have several.

The purpose of the `print_station_extremes` function is to take an array of `Observation` objects and print out a summary of the **extreme observations** seen at each station. The extreme observations are the minimum observed temperature (and the time of that observation) and the maximum observed temperature (and the time of that observation). In the event that the same extreme is observed multiple times (for example, if the minimum observed temperature is -6.10 degrees, but was observed on two separate occasions), the program will report the date and time of the chronologically earliest occurrence of that temperature in the input (which will **not** necessarily be the first such observation to appear in the input file, since the records in the input file may appear in any order).

The output will have one line for each station ID in the input dataset. The output will be formatted to be human readable, using the formatting shown below (the exact `printf` formatting string to use is given in the specification in the header file). The output must be printed **ordered by station number**. Stations for which no observations were recorded must be omitted from the output. Stations with at least one observation must appear exactly once (it is not permitted for multiple lines of output to refer to the same station). For example, on the set of observations taken from the input file shown above, the `print_station_extremes` function would print the following output (and **nothing else**).

```
Station 1: Minimum = 7.30 degrees (2020-11-13 18:06), Maximum = 8.70 degrees (2020-11-14 12:07)
Station 2: Minimum = -1.00 degrees (2020-12-01 01:06), Maximum = -0.50 degrees (2020-12-01 09:07)
Station 111: Minimum = 18.70 degrees (2020-06-10 17:00), Maximum = 18.70 degrees (2020-06-10 17:00)
Station 143: Minimum = 8.60 degrees (2020-11-14 11:58), Maximum = 8.60 degrees (2020-11-14 11:58)
Station 163: Minimum = 8.90 degrees (2020-11-14 13:03), Maximum = 8.90 degrees (2020-11-14 13:03)
```

4 Evaluation

Your code must compile and run correctly in the CSC 111 virtual lab environment using the compile and run commands given above. If your code does not compile **as submitted** (with no modifications whatsoever) when combined with the provided `temperature_data.h` file and a suitable main program, you will receive a mark of zero. Note that you are not permitted to change anything in the `temperature_data.h` file, since only the `temperature_data.c` file will be collected for marking, so you must make sure that your function definitions comply with the specification in the provided version of the `temperature_data.h` file.

This assignment is worth 3% of your final grade and will be marked out of 17.

Second Chance Submission

After the initial due date (Sunday, Dec. 6th, 2020) has passed, your code will be automatically tested using a variety of inputs (which will not necessarily match the example inputs shown in the previous sections). If your programs do not give correct output on some of the tested inputs, you will be allowed to fix your program and resubmit it until Wednesday, Dec. 9th, 2020. The grade for the assignment will be computed to be the average of the grade for both submissions (so you can recover some, but not all, of your marks by fixing and resubmitting your code). If you do not resubmit your code, your initial submission will be used in place of the resubmission.

If you do not submit an initial version of your code before the due date, you are **not** permitted to resubmit your code and will receive a mark of zero.

Submission Instructions

All submissions for this assignment will be accepted electronically. Submit your `temperature_data.c` file through the Assignments tab on conneX. You are permitted to delete and resubmit your assignment as many times as you want before the due date, but no submissions will be accepted after the due date has passed. For the second chance submission described above, a separate assignment entry will be created after the initial due date.

Ensure that each file you submit contains a comment with your name and student number, and that the files for each question are named correctly (as described in the question). If you do not name your files correctly, or if you do not submit them electronically, it will not be possible to mark your submission and you will receive a mark of zero.

After submitting your assignment, conneX will automatically send you a confirmation email to your `@uvic.ca` email address If you do not receive such an email, your submission was not received. If you have problems with the submission process, send an email to your instructor **before** the due date.

To verify that you submitted the correct file, you can download your submission from conneX after submitting and test that it works correctly in the virtual lab environment. It will be assumed that all submissions have been tested this way, and therefore that there is no reasonable chance that an incorrect file was submitted accidentally, so no exceptions will be made to the due date as a result of apparently incorrect versions being submitted.