Lab 11
Weather stats

7 November 2019

This lab will give you continued practice working with structs and with vectors and with vectors of structs (and writing functions for them, and additionally will involve reading in data rather than hand-building the vectors and structs in a test suite.

1. To get started on this part of the lab, copy some files into your directory. The task of the week will involve processing weather data, so first, you'll copy a set of weather data, found in

   ```
   /home/shared/160/weather-big.txt
   /home/shared/160/weather-small.txt
   ```

   into your own working directory for this lab.

2. Look at (but don't change) the contents of the files, then continue reading for their description.

These are data about the weather for the month of August 2009 in Galesburg, IL (where I worked at the time). Each line of the file contains information about the date and time it represents, as well as the temperature (in degrees Fahrenheit) and the wind speed (in miles per hour). This is a sample line illustrating the format:

```
8  2 2009      18 00   76      12
```

The date is first, then the time: this line represents August 2, 2009, at 18:00 (6pm). The temperature was 76°F, and the wind was blowing at 12mph. The "big" file contains measurements for every hour that month; the "small" file contains two days. (You're also encouraged to create your own files, even smaller, for testing purposes, but make sure they follow the same format.)

3. Based on the description above, and using the examples of `FullName` and `GroceryItem` and `Location` (and the struct you wrote for last

week's lab), define a `struct` called `Weather` that is capable of holding all the data on each line of our data file. This definition should go in a file `Weather.h` (which will also be where the related function headers will eventually go).

4. Create a file `test_Weather.u`, make a test suite with only (for now) a `fixture` section, and in that section make at least two examples of `Weather` objects, and one `vector` that contains those two `Weather` objects.

5. Compile it! With just the `.h` and the fixture in the `.u` there is enough to compile. No tests to run yet, but you can verify if your syntax is correct.

6. Create a file called `run_weather_stats.cpp` that contains a `main` function, making sure to `#include` the `.h` file you just created. Compile it to check your syntax.

7. In that `main`, read in all the pieces from one line and make a `Weather` value that bundles them up. (There are seven pieces on the line, all of them integers.) (Look at the files in `/home/shared/160/1106-10/` or `.../1106-11/` to see what we defined in class, for a pattern to follow.)

8. Verify that it compiles. Have you started a readme yet to put the compiling and testing and running commands into? You should start a readme to put the compiling and testing and running commands into.

9. Still in `main`, wrap that in a loop, as we did in class Wednesday, that keeps reading until we run out of input, and builds a vector of `Weather` objects.

10. Add a function header to `Weather.h` for a function named `everBelow55` that will determine whether the temperature in a given vector of weather values ever dipped below 55 degrees.

11. Create a file `Weather.cpp` with a stub for the function you designed in the previous step.

12. Go back to the file `test_Weather.u` and add a `tests` section to your suite, with tests for the `everBelow55` function. Use the vector that you already put in the fixture, and add another vector so that you can more thoroughly test the function.

13. Compile again—note that your compile line will need to include both the `.u` file and the newly made `Weather.cpp`, or you'll get a linker error. (Don't forget to update the readme!) Remember that it's ok that the tests are failing now (and if they're not, they're broken tests, and you should resolve that).

14. Go back to the `run_weather_stats.cpp` file and after the vector of weather values is read in, have it print the result of a call to the function you've been writing. (Note that `cout` prints `bool` values as 1 and 0 by default, which is fine by me if it doesn't bother you.)

15. Finish writing the function.

I'll be circulating around the lab to answer questions. If you're stuck on some part of the drill, ask me about that (and while you're waiting for me to get to you, look at the next section about vim commands). If you're not stuck but haven't finished the drill, work on that now. If you're done with the drill, go on to the next section.

Vim FOTD: Miscellaneous commands

A few commands you might find handy. Remember, all commands are case sensitive and work in command mode.

`J`	Join this line and the next
`rX`	Replace this character with X, for any value of X

Commands that initiate insert mode:

`s`	Replace this character
`cc`	Replace this whole line *
`I`	Insert at beginning of line (but after whitespace)
`a`	Insert after this character
`A`	Insert at end of line
`o`	Add a line after this one, and insert there
`O`	Add a line before this one, and insert there
`==`	Reindent this line *
`<<`	Shift this line to the left *
`>>`	Shift this line to the left *
`.`	Repeat the previous command

All commands marked with a * are given in "this whole line" form, but (like `dd` and `yy`) can also be used with any movement command, so for instance

`cw` replaces from here to the end of this word with whatever is typed in insert mode, `=G` reindents from here to end of file, and `>%` shifts to the right everything between this line and the matching curly bracket.

Also, these commands and every single other command in vim can be preceded with a number (can be more than one digit), which typically means "repeat N times", though the exact interpretation varies from command to command. `15G` will go to line 15 of the file, for example, and `3rQ` will replace the next three characters with the character `Q`.

Functions to process vectors of weather

For this part of the lab, write functions that take vectors of `Weather` and process them. In some cases the result would be a single number; in others it could be a `Weather` value or even a whole vector of `Weather` values.

- A function `average_temp` that computes the average (mean) temperature of all the `Weather` values in the given vector.
- A function `min_temp` that computes the minimum temperature of all the `Weather` values in the given vector.
- A function `max_wind` that computes the maximum wind speed of all the `Weather` values in the given vector.
- A function `coolest_time` that finds and returns the `Weather` (which includes date/time info) that represents the coolest moment in the given vector. (If there are multiple equal "coolest" moments, any one of them could be returned.)
- A function `noon_data` that filters and returns a vector of all those `Weather` values in the given vector that represent a measurement taken at noon on some day.

In each case, the function header should be typed in to `Weather.h` and then the function should be defined in `Weather.cpp` and called in `run_weather_stats.cpp` to print their result. You should also clearly test each function you implement, in the `.u` file.

Don't forget that we've written functions that are very similar to these before! Feel free to refer back to earlier labs and classwork.

Handing in

Hand in as `lab11`; it is due Wednesday at 4pm as usual.

RUBRIC

General

1 Attendance at lab with drill done or question written down

1 Documentation clear and correct on how to compile, run, test

Drill

1 Valid and correct `struct` definition, files compile and run

1 Reads weather data from `cin` into `vector` of `Weather`

1 Function `everBelow55` tested and defined correctly

Other Weather functions

1 Headers and test cases for min, average temperature and max wind functions

1 Definitions for min, average temperature and max wind functions

1 Computes coolest time using function (+ tests)

1 Computes list of noon-time data using function (+ tests)

½ Prints output of running defined functions on input

½ Tests pass OR there is a note in readme