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.