Lab 11
Sorting

17 November 2015

This week we'll continue working on the idea of putting values in order—sorting—and what has to be done to accomplish this. The "drill" is getting some bookkeeping out of the way so that we can dive right into the sorting algorithms during lab.

Drill: *Verifying* sortedness

1. Start files `sorting.h` and `sorting.cpp` with a function `is_in_order` that takes a `vector<string>` parameter and is intended (eventually) to determine whether the given vector is in sorted order or not. For now, though, just have it always return `false`. Make sure this much compiles by running

   ```
   compile -c sorting.cpp
   ```

   which, again, runs the compiler without running the linker (since we haven't defined a `main` function yet). (As *always*, try to compile as often as is reasonable, and start running your code as soon as it's feasible to do so.)

2. Write another program file `run_sort.cpp` that has a `main` function that reads lines from `cin` as long as there are any to be read, and puts them in a vector; and once it's done reading them, it calls `is_in_order` on that vector and prints the result. (Remember that that function is so far only returning false, and false prints as zero, so it's ok and expected that your program will just print zero regardless of its input!)

3. Start a unit test file `test_sorting.u` that will test whether `is_in_order` works correctly. Use the `test_stat_functions.u` file from Lab 6 as a model (on page 4 of the lab handout)—in this case the vector should be `vector<string>` rather than `vector<double>`, and should contain examples of `string` rather than examples of `double`, and you should have examples of vectors that *are* in order and vectors that *aren't* in order, to effectively test the function. In the actual test cases, you'll have lines that look something like this:

```
check (is_in_order(vec4)) expect true;
```

assuming you have a vector named `vec4` (you should probably pick a better name) and it is already in order. If it weren't in order, you would `expect false` instead.

If you have test cases that are meaningfully different, you can put them in separate `test` blocks if you like. Just make sure they're all part of the same `test suite` block (there should be only one of those for the whole file).

4. Return to `sorting.cpp` and edit `is_in_order` to do a little bit of work. If the given vector is of size 1 or shorter, immediately return true. If it's longer, compare (using the regular `<=` operator) the values of the first two elements of the given vector, and if the second is smaller than the first, return false; otherwise return true. Compile and run your test cases. Do they all succeed at this point? Should they? If all your tests succeed, continue to step 5. If any fail, continue to step 6.

5. If all your tests succeed in spite of the fact that you know your implementation of `is_in_order` is incomplete (it only compares two items), then that means your test cases aren't comprehensive enough. Add a test case that will catch this incompletely-implemented function; and compile it and run the test and see the test fail, indicating that `is_in_order` isn't done yet. Then continue to step 6.

6. At least one test is failing, so you have more work to do with `is_in_order`. Specifically, loop through indices and compare *every* adjacent pair in the vector to see if they're out of order. If you find an out-of-order pair, return false. If you get to the end of loop without finding an out-of-order pair, then you can return true.

7. If at any point you run the tests and they pass even though you're not done, go back to step 5: add a test case to verify whatever you forgot to check.