Lab 10
Structs

31 October 2019

This week we practice creating our own `struct` type and writing some functions that process it. The context will be (we imagine) that you are writing a blog or social media site where users can post messages, and need to store user information; your struct will represent one user, and store their login name, their email address, and the number of posts they have made.

For the drill, we'll build the type and write and test one function that operates on that type.

1. First, start a file `User.h` and in it, define a `struct` with three fields, to represent the login name, the email address, and the number of messages posted (in that order). Include a comment above the struct to describe what the purpose of the data type is.
2. Below the `struct` definition, add a declaration for the function `hasPosted`, which determines whether a given user has ever posted a message. Include a comment above the declaration with that description.
3. Start a file `User.cpp` with the appropriate `#include` and with a stub definition for the `hasPosted` function.
4. Start a file `test_User.u` with some appropriate tests for the `hasPosted` function.
5. Edit your readme to add instructions on how to compile and run the tests for the functions you'll be writing (and also with the other stuff that needs to go in documentation, if you haven't already!).
6. Go back to the `.cpp` file and fill out the body of the `hasPosted` function.

Remember to look at the posted chapter as a reference; also, remember that from time to time I post the photos of the whiteboard that I take at the end of each class period—you should be able to mine those for information on how to work with structs. Also see the files in `/home/shared/160` .

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 movement). If you're not stuck but haven't finished the drill, work on that now. If you're done with the drill, continue on to the next section.

Vim FOTD: help!

Edit a file (such as one of your User files) using `vim` and type

```
:help
```

and hit enter. The session subdivides into an extra window, which has some helpful text in it. Now type

```
:help dd
```

and see what it says. In general, any command-mode command can be explained in this fashion. Likewise colon-mode commands ("`:help :w`") and command-line options ("`:help -o`") and even settable configuration options that you put in your vimrc ("`:help 'incsearch'`").

To close the extra help window inside of Vim, use `:q` just like you normally do to close a file—it will return the file you were editing to filling the entire PuTTY window.

It turns out that Vim is a pretty vast program, with an immense number of features. You'll keep discovering them, and any time you hear about a feature or suspect its existence, there's a good chance you can find out more about it using the `:help` system.

More functions on `Users`

Write `sameName`, which determines whether a given `User`'s login name is the same as their email login (the part before the `@`). (Hint: the `find` function of a string that we saw in class might be handy...)

Write `emailDomain`, which computes the domain (the part after the `@`) of the email address of a given `User`.

Write `makeUser`, which builds a new `User` value with given login name and given email address. (What should its initial number of posted messages be?)

Write `includesUser`, which determines whether a given vector of `User`s includes one with a given login name.

Write `countNewbies`, which counts the number of users in a given vector of `User`s who have never posted a message.

Handing in and rubric

Hand in as `lab10`. Due 4pm next Wednesday.

RUBRIC (Tentative)

1 Attendance at lab with drill done or question written down
1 Appropriate documentation throughout (readme, doc comments)
Drill (`hasPosted`)
1 Valid and correct `struct` definition, files compile and run
1 Test case file set up, compiles, good TC for hasPosted
1 hasPosted is defined correctly
Rest of lab
5 sameName, emailDomain, makeUser, includesUser, countNewbies
Each:
1/2 correct header, compiles, good test cases
1/2 correct definition

As before, remember that some points are available for good test cases *even or especially* if those test cases are not passing.