

Lab 10

Structs

3 April 2025

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.

To begin with, 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 `has_posted`, 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 `has_posted` function.
4. Start a file `test_User.u` with some appropriate tests for the `has_posted` 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 `has_posted` function.

Remember as always to refer to the course pack, posted photos and videos, and the in-class examples in `/home/shared/160/` to help you figure out what to do!

More functions on Users

Write `same_name`, which determines whether a given `User`'s login name is the same as the name part of their email (the part before the `@`). (Hint: while you could develop your own helper function to do this, the `.find` method of a string would be useful here. How would you get information on how to use it? Also remember that you can write test cases even before you figure out how to correctly implement something!)

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

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

Write `includes_user`, which determines whether a given vector of `Users` includes one with a given login name.

Write `count_newbies`, which counts the number of users in a given vector of `Users` 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 preview done or question written down
- 1 Appropriate documentation throughout (readme, doc comments)
- Part 1 (has_posted)**
- 1 Valid and correct `struct` definition, files compile and run
- 1 Test case file set up, compiles, good TC for `has_posted`
- 1 `has_posted` is defined correctly
- Rest of lab**
- 5 `same_name`, `email_domain`, `make_user`, `includes_user`, `count_newbies`
Each:
 - $\frac{1}{2}$ correct header, compiles, good test cases
 - $\frac{1}{2}$ correct definition

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

AI policy and frequent submission

(no substantive change from the Lab 3 version of this policy)

Some use of generative AI is fine, but you a) should not paste this assignment or type it verbatim into the AI prompt, and b) should not be asking the AI for the whole program all at once. (Just like you can ask for help from a human, but should not have them write the whole program for you!) If you get help from an AI chat OR from a person, you should note that in a program comment near whatever you got from them.

Relatedly, I expect that you'll run the `handin` program relatively often, and I *require* that you do so at least 2–3 times over the course of working on the lab. As a rule of thumb, hand it in after completing each 1–2 points on the rubric. Submissions that jump straight to a final, (near-)correct version with no intervening submissions along the way *will receive little or no credit* for that part.

This is a new policy that I'm experimenting with; let me know if you have any feedback.

This document was written and prepared without the use of generative AI.