Lab 3 - C programming; remote logins

Due Friday, September 13, 2019

Setup: Log into your personal account on your Pi, and create a subdirectory for this lab

```
cd HD
mkdir lab3
cd lab3
```

A Example of C programming

In-class discussion of the `example.c` program

- Some C syntax diagrams

Exercises

1. Copy the `example.c` program to your `lab3` directory by one of these methods:
   - Download `example.c` from the link above onto your computer, then copy it to your pi using `scp` **on your computer** (works for Mac and Linux computers or WSL terminals)
     ```
     scp ~/Downloads/example.c 10.0.0.254:HD/lab3
     ```

   OR

   - **On your Pi,** retrieve the `example.c` program from its URL using the `wget` program
     ```
     wget https://www.stolaf.edu/people/rab/hd/pub/example.c
     ```
2. Compile and run `example.c` on your Pi.
   ```
   gcc example.c -lm -o example
   ./example
   ```
   Be sure to include the `-lm` option for `gcc`, which loads the math library code (needed for the `sqrt()` function).
3. Create a commit of your initial version of `example.c`.
   ```
   git add example.c
   git commit -m "Lab 3 first version of example.c"
   ```
4. Use the `cp` command to make a copy of `example.c` called `example1.c`, and change it to initialize `i` at `120` instead of `7`. Compile and run the modified code, and examine the results. It this what you expected?
5. Write a program `sqrt0.c` that prints the square root of a floating-point number `x`.
   - Define the number `x` as a float variable in your `main()`.
     - **Note:** we will examine how to enter values interactively in C in the next lab.

   - Then, print `x` and its square root, labelled on one line (compare to `example.c`).

   This is a short program to test a few of the concepts in `example.c`.

6. **Carry this step out on paper.** *Attempt to get the C syntax for program code exactly correct, including semicolons, etc.*
   - Write a C language statement that prints whether a floating-point variable `v` is positive, zero, or negative. Include the value of `v` in your output.
     - For example, if `v` has the value `-7.3`, your statement should print
       ```
       -7.3 is negative.
       ```
     - Use a ***nested*** `if` to separate the three cases. The final statement of [`example.c`](example.c) (just before `return i;`) is an example of a nested `if`.
     - To compare `v` with zero, use the ***C equality operator*** `==`. For example,
       ```
       x == 5
       ```
       is `true` if `x` has the value `5` and `false` otherwise.
       - **Warning: Use only == when you want to compare two values in C programs.** A single `=` is the ***assignment operator***, and the expression `x = 5` would *change the value of* `x` to `5`, instead of determining whether the existing value of `x` is the same as `5`.
       - Using `=` instead of `==` when you want to compare values is a very easy mistake to make - watch for it.
   - Write a guard to test whether a float variable `f` has an integer value
     - For example, `4.0` has an integer value, but `4.01` does not.
     - A ***guard*** is a boolean expression (evaluates to either `true` or `false`); we use the term guard for boolean expressions used to decide something in programming. For example, the expression `i < 10` is a guard in the C-language statement
       ```
       if (i < 10)
         printf("%d is small\n", i);
       ```
     - To determine whether the value of `f` is an integer, find the ***integer conversion*** of `f` and see if that conversion is equal to the original value of `f`. The integer conversion of `f` may be found using this syntax:
       ```
       (int) f
       ```
       For example, if `f` has the value `3.14`, then `(int) f` has the value `3`.
   - Write another guard that tests whether the square root of an integer variable `i` is itself an integer. For example, your guard should return `true` when `i` has the value `16` or `25`, but `false` if `i` has the value `17`. You may assume that `i` has a non-negative value for this problem.

   Now, check your three answers, and try to find any errors in syntax or computation to verify that they are correct.

7. Write a program `tests.c` that checks all three of your by-hand answers to the previous problem. Compile and run your program and verify that your answers and your program are correct; fix any bugs.

   Notes:

    - For example, in the first part of `main()` for `tests.c`, define a float variable `v` with a value of your choice, followed by your first answer above. When you run your program, the output should correctly identify whether `v` was positive, zero, or negative.
    - Proceed to write a test for your second answer's guard expression, by defining a float variable `f` with an initial value of your choice, then writing an `if` statement using your guard to print a message such as
      ```
      4.3 does not have an integer value
      ```
      or
      ```
      4.0 has an integer value
      ```
      depending on the value of `f`.
      Also test your third answer (another guard).

   Note: If you had to change your hand-written expression(s), indicate the part that needed to be changed on your paper. (Identifying a mistake won't cause any point loss for this assignment -- writing the change is intended to help you learn better.)

8. Write a second test program `tests2.c` that checks all the other cases that `tests.c` didn't check. Compile, run, and fix any errors.
    - For example, if your test value for `v` happened to be positive, then check that your nested `if` statement also prints the expected output when `v` is zero, and when `v` is negative. This will require writing your nested `if` statement twice within `tests2.c`.
        - Note: Include only one definition of the variable `v` in `tests2.c`, and *assign* a different value to `v` between your two nested `if` statements in `test2.c` in order to change `v`'s value.
    - You can use the same check statements for the two guards in `tests2.c` as you did in `tests.c`, except to change the values of `f` and `i`.
        - **Optional/extra:** Add a check for the assumption that `i` is non-negative to `tests2.c`, and verify that your check works.

9. **To submit your work in this part:** carry these steps out on your Pi.

```
cd ~/HD/lab3
git add example.c example1.c sqrt0.c tests.c tests2.c
git commit -m "lab3: Part A complete"
git pull origin master
git push origin master
```

Note: if your work is not yet complete, use a different commit message indicating what you have done, e.g., `"lab3: Part A step 1 only"`

B Logging into a Link computer; passwordless SSH

1. Log into your pi (from your laptop or a Link computer), using your personal account.
2. SSH from Pi to a (different) Link machine, e.g., rns202-1.cs.stolaf.edu

   `ssh` *`username`*`@rns202-1.cs.stolaf.edu`

   Supply your St. Olaf password (same as for your email).
   Note: You can use any Link computer, not necessarily rns202-1, and it's a good idea to avoid everyone using the same link computers in general. Here's a complete list of Link machines (some may be "down" at a given time):
   - rns202-1.cs.stolaf.edu to rns202-21.cs.stolaf.edu
   - rns203-1.cs.stolaf.edu to rns203-16.cs.stolaf.edu

   Enter commands such as `ls` to explore, then log out using `exit`

3. *Passwordless SSH* is a mechanism that enables you to log in to a computer without supplying a password. It is actually more secure than providing a password, since your password is not transmitted over the network (which someone could be snooping…).

   We will now set up passwordless SSH between your Pi and the Link computers. This requires two steps: Creating an *SSH Key pair*; and *copying the public key to a Link machine*.
   Note: It's enough to set up passwordless SSH to a single Link computer, since all the Link computers share your same home directory for your IT account.
   - After HW1, you already have an SSH key pair: it was created when you logged into your Pi and entered

     `ssh-keygen`

     An *SSH key pair* consists of two "key" files, one of which should be kept secret, and one of which can be shared publicly anywhere. The combination of these two files makes it possible to communicate securely over a network using *public-key encryption*, as long as you keep your secret key private.
   - Now (still logged into your Pi) enter the following command:

     `ssh-copy-id rns202-1.cs.stolaf.edu`

     (Any Link machine may be chosen instead of rns201-1)
     This command will prompt you for your password on that Link machine, so it can install the *public key* file on your IT account on that Link machine (and hence all Link machines).
     This should set up passwordless SSH from your Pi to your Link machine account.

To test this step: While logged into your Pi, enter `ssh` *`username`*`@rns202-1.cs.stolaf.edu` for some link machine (not necessarily rns202-1). You should log in automatically, without having to supply a password!

Log out from that Link machine using `exit`. Then try to SSH into a different Link machine than the one you just logged into, and verify that you can log in to that Link machine without a password, too. Then `exit` from that second Link machine.

4. As mentioned above, the command `ssh-keygen` creates a pair of files that are used for securely transmitting messages over a computer network.
   - The *public key* can safely be shared with other *remote* computers (e.g., link machines), and can be used on those remote computers to *encrypt* messages (convert them into a secret code).
   - The *private key* is kept secret on the local computer (e.g., your Raspberry Pi), and can be used to *decrypt* those messages.

   These keys are long strings that would be practically impossible to guess or deduce, without knowing those keys. Since any computer with the public key can encrypt a message but only a computer with the private key can decrypt that message, this *public-key encryption* system enables computers to communicate securely with each other.

   The command `ssh-copy-id rns202-1.cs.stolaf.edu` copies the *public* key (only!) to the destination computer rns202-1.cs.stolaf.edu, thus setting the stage for secret communication. Finally, the `ssh` command uses this secret message system to convince a remote machine (e.g., Link machine) with the *public key* that your user account on your local computer (e.g., your Pi) possesses the corresponding *private key,* so it's safe for your account on your local computer to receive a login without a password.
   - *Some optional extra information:*
     - The remote machine believes your account doesn't need a password because you already entered your correct password during the `ssh-copy-id` command.
     - Copying your public key to just one Link machine actually grants you access to all of them, because all the Link machines share the same home directory for your Link user account.

   Note that **passwordless SSH is more secure than using SSH with a password!** This is because logging in over a network using a password requires transmitting that password over the network, which raises the risk of someone else on the network somehow discovering that password. But public-key encryption does not carry this risk.
   - This is rare in computer security - **the more convenient way to login is actually more secure!** Usually, you have to give up some convenience in order to get better security (for example, using a difficult-to-guess password is more secure, but less convenient).

DO THIS:

1. Copy your *public key* into your `lab3` subdirectory **of your Pi** as follows:
   `cp ~/.ssh/id_rsa.pub ~/HD/lab3`
2. Now submit that public key. **Note:** This is safe for the public key, but **not** for the private key.

```
cd ~/HD/lab3
git add id_rsa.pub
git commit -m "lab3: Part B complete"
git pull origin master
git push origin master
```

Deliverable files for Lab 3 (in `~/HD/lab3`): `example.c example1.c sqrt0.c tests.c tests2.c id_rsa.pub`