CSE 333 Homework 0

Out: Thursday, September 25, 2025
Due: Wednesday, October 1, 2025 by **11:59 pm**.

Summary

The goal of Homework 0 is to make sure that all of the course infrastructure is working for you:

1. You will fetch a source distribution that we have prepared.
2. You will trivially modify a simple hello world C application, use `make` to compile it, and run it to learn the magic code.
3. You will run tools to check the code for memory issues and potential style problems.
4. You will package up and submit your code.

Homework 0 Instructions

Part A – Fetch the Code

Assignments will be distributed and collected using your CSE 333 Gitlab repository. Starter code will be added to your repository by the course staff and you will tag your repository when you are done with an assignment to indicate which revision should be evaluated and graded by the course staff. More information about git and Gitlab is given below and in other writeups on the course web site.

Clone your Gitlab repository

You can use the CSE Gitlab web interface (https://gitlab.cs.washington.edu) to browse your files and find out information about your repository, but you must clone a copy to a Linux machine to do your work. If you have not done so already, change to a directory where you want to store the local repository for your CSE 333 projects, then follow the instructions on the Gitlab Setup page (../../gitlab/#setup) to clone your CSE 333 repository. There is also additional information further down that page on Developing with Git Workflow (../../gitlab/#workflow).

You might have additional personal projects and repositories in Gitlab for your own work, but be sure to use the repository provided by us for your CSE 333 course projects.

Once you have cloned the repository, change into that directory (it will be named something like `cse333-25au-xyzzy`, where `xyzzy` is your userid) then enter a `ls` command. You should see something like this:

```
$ git clone git@gitlab.cs.washington.edu:cse333-25au-students/cse333-25au-xyzzy.g
-- git output appears here
$ cd cse333-25au-xyzzy
$ ls -a
.gitignore cpplint.py exercises hw0
```

- The `.gitignore` file lists file types that should not normally be saved in the repository. These are typically things like editor backup files (names ending in `~`) and object files (ending in `.o`). They are files that are generated and used locally while you are working but are recreated as needed from files that are in the repository rather than archived permanently.
- The `exercises` directory is mostly empty, but you can store your exercise code here. It is especially useful for enabling course staff to access your code remotely (*e.g.*, during virtual OH or when responding to your Ed post). It these cases, it would be best to push your current code to this folder for a staff member to view.

If you originally cloned your repository before the staff added the `hw0` directory to it, enter a `git pull` command to bring your local copy up to date. Once you see the `hw0` directory, enter `cd hw0` to change into that directory to work on the assignment.

Part B – Edit, Compile, and Run `hello_world`

In the `hw0` directory, run `make`. This command will use the instructions in file `Makefile` to compile the `hello_world` executable using the `gcc` compiler. Run the executable by entering the command `./hello_world`. You should see one line of output that looks like:

```
$ ./hello_world
The magic code is: <xxxxx>
$
```

for some <xxxxx>. Now, edit `hello_world.c` in the CSE Linux environment using your favorite editor (../../editors/), and change the `printf` so that it instead says:

```
$ ./hello_world
The magic word is: <xxxxx>
$
```

Execute `make` to rebuild, and then re-run `./hello_world` to make sure that it does what you expect.

Part C – Try GDB

You're unlikely to have any runtime errors, but let's use the debugger (http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html) so that you know what it can do. Here is a session that:

1. Launches the debugger,
2. Sets a breakpoint on source line 18 of `hello_world.c` ,
3. Prints the values of a couple of variables and an expression,
4. Prints a "backtrace" of the stack (showing the sequence of procedure calls that led to executing line 18 of `hello_world.c`), and then
5. Continues execution.

You should type the lines shown in italics. You may see some differences in version numbers or exact addresses, but the data values should be basically the same.

- For GDB to be useful, you must use the -g option for `gcc` when compiling. The `Makefile` we distributed does that.
- If you have already modified your file for Part B, you will see the new output word instead of "code" below.

```
$ gdb ./hello_world
GNU gdb (GDB) Red Hat Enterprise Linux 10.2-10.el9
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
...
Reading symbols from ./hello_world...
(gdb) break 18
Breakpoint 1 at 0x401143: file hello_world.c, line 18.
(gdb) run
Starting program: /homes/iws/<netid>/cse333-25au-<netid>/hw0/hello_world
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffe1c8) at hello_world.c:18
18        printf("The magic word is: %X\n", a + b);
(gdb) print argv[0]
$1 = 0x7fffffffe477 "/homes/iws/<netid>/cse333-25au-<netid>/hw0/hello_world"
(gdb) print a
$2 = -889262067
(gdb) print /x a
$3 = 0xcafef00d
(gdb) print a+b
$4 = -559038737
(gdb) backtrace
#0  main (argc=1, argv=0x7fffffffe1c8) at hello_world.c:18
(gdb) continue
Continuing.
The magic word is: <xxxxxxxx>
[Inferior 1 (process 396554) exited normally]
(gdb) quit
$
```

Part D – Verify No Memory Issues

Throughout the quarter, we'll also be testing whether your code has any memory leaks or other memory errors (*e.g.*, using uninitialized memory). We'll be using Valgrind (https://valgrind.org/) to do this. Try out Valgrind for yourself so you know how to run it:

```
$ valgrind --leak-check=full ./hello_world
```

Note that Valgrind will print out that no memory issues were found. Specifically, look for a line that looks like:

```
==399152== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Part E – Check for Style Issues

Another requirement during the quarter is that your code must follow the provided style guidelines. Although there are many opinions about what constitutes "good style," in this course, we will generally follow the Google C++ Style Guide (https://google.github.io/styleguide/cppguide.html) for both C and C++ code. The repository files for this assignment included a Python script `cpplint.py` to check C source files for style issues. Try it yourself so you know how to run it:

```
$ ../cpplint.py --clint hello_world.c
```

No style-checking tool is perfect, but you should try to clean up any problems that `cpplint` detects unless it flags something that is definitely *not* a problem in this particular context (but be sure you have very good reasons to ignore any warnings).

Part F – Add a `README.md` File

Create a `README.md` file in directory `hw0` that contains:

- Your name
- Your UW or CSE email address
- A sentence that reveals the magic word you learned from Part B. Be sure that you actually say the magic word as it was output from the executable; do not just hint at the word or its meaning.

We are not picky about the format of this file as long as it has all this information.

Homework Submission

Once you're done, "turning in" the assignment is done by creating an appropriate tag in your git repository to designate the revision (commit) that the course staff should examine for grading. But there are multiple ways to get this wrong, so you should *carefully* follow the following steps *in this order*. The basic idea is:

1. Tidy up and be sure that everything is properly committed *and pushed* to your Gitlab repository.
2. Add a tag to your repository to specify the commit that corresponds to the finished assignment, *after* you have pushed all of your files.

3. Check out a fresh copy of the repository *in the CSE Linux environment* and verify that everything works as expected.

Tidy Up and Commit

Commit all of your changes to your repository (see the beginning of the assignment or the main course web page for links to `git` information if you need a refresher on how to do this). Then in the `hw0` directory:

```
$ git pull
$ make clean
$ git status
On branch main
Your branch is up-to-date with 'origin/main'.
nothing to commit, working directory clean
```

If you see any messages about uncommitted changes or any other indications that the latest version of your code has not been pushed to the Gitlab repository, fix those problems and push any unsaved changes before going on. Then repeat the above steps to verify that all is well.

Tag Your Repository

Tag your repository and push the tag information to Gitlab to indicate that the current commit is the version that you are submitting for grading:

```
$ git tag hw0-final
$ git push --tags
```

Do not do this until *after* you have pushed all parts of your homework solution to Gitlab.

Check a Fresh Copy

To be sure that you *really* have updated everything properly, create a *brand new,* ***empty*** directory that is *nowhere near* your regular working directory, clone the repository into the new location, and verify that everything works as expected. It is really, *really,* ***REALLY*** important that this not be nested anywhere inside your regular, working repository directory.

```
$ cd <somewhere-completely-different>
$ git clone git@gitlab.cs.washington.edu:cse333-25au-students/cse333-25au-xyzzy.g
$ cd cse333-25au-xyzzy
$ git checkout hw0-final
$ ls
cpplint.py exercises hw0
```

Use your own userid instead of `xyzzy`, of course. The commands after `git clone` change to the newly cloned directory, then cause git to switch to the tagged commit you created in Step 2 above. We will do the same when we examine your files for grading.

At this point you should see your `hw0` directory. `cd` into it, run `make`, run any tests you wish (something that will be crucial on future assignments). If there are any problems, *immediately* erase this newly-cloned copy of your repository (`rm -rf cse333-25au-xyzzy`), go back to your regular working repository, and fix whatever is wrong. It may be as simple as running a missed `git push --tags` command if the tag was not found in the repository. If it requires more substantive changes, you may need to do a little voodoo to get rid of the original `hw0-final` tag from your repository and re-tag after making your repairs.

To eliminate the `hw0-final` tag, do the following (this should not normally be necessary):

```
$ git tag -d hw0-final
$ git push origin :refs/tags/hw0-final
```

Then make and commit *and push* your repairs, and *only* after all the changes are pushed, repeat the tag and tag push commands from Step 2. *And then repeat this verification step to be sure that the updated version is actually correct.*

Note: if you discover that repairs are needed when you check your work, it is *crucial* that you delete the newly-cloned copy and make the repairs back your regular working repository. If you modify files in the cloned copy you may wind up pushing changes to Gitlab that will leave your repository in a strange state, and files may appear to mysteriously vanish. Please follow the instructions precisely.

Grading

For hw0, we'll give you full credit if you learn the magic code/word and correctly submit your work.