Advanced - Make (Build Systems)

EECS 201 Winter 2025

Submission Instructions

This assignment will be submitted as a repository on the EECS GitLab server (see Basic - Git 1). Create a private, blank, **README-less (uncheck that box!)** Project on it with the name/path/URL `eecs201-adv-make` and add `brng` as a Reporter. The submission branch will be `build`. If this branch is not already the default initial branch, you initialize the local repo with an additional argument: `git init --initial-branch=build` or `git init -b build` if your version of Git is recent enough. Otherwise you can create a branch with this name after your first commit.

To finish the submission process and get your grade, you will need to run the autograder. You will need to SSH into the course server (see Basic - Intro):
`local$ ssh uniqname@peritia.eecs.umich.edu`
(the `local$` referring to a shell prompt on your system)

and then run `eecs201-test`:
`peritia$ eecs201-test adv-make`
(the `peritia$` referring to a shell prompt on the server)

```
/
|-- report.txt
|-- Makefile
|-- CMakeLists.txt
```

Preface

First, initialize a new Git repository and set up its remote appropriately. This repository will simply be used as a mechanism to submit a few files for this assignment.
In this assignment you'll be provided yet another zipped archive containing some starter files.

`https://www.eecs.umich.edu/courses/eecs201/wn2025/files/assignments/adv-make.tar.gz`

These starter files are not to be submitted in your repository. They just serve as baseline data for you to work with.

1 A more featured Makefile (7)

In this part you'll be writing a more featured Makefile that extends some behavior of Makefiles written in the basic assignment, now with the addition of an object code and a linking step.

- This Makefile is meant to sit inside of `example-project` when run. Keep in mind the repo is mainly for gathering files for submission, and how you test the Makefile can be independent from how you submit it.
- All source code files will be in the `src/` directory
- Additional header files for the include path will be in the `inc/` directory. You will need to figure out how to add this directory to the include path for your compilation steps.
- `example-project` is only an example provided to you. When graded, there may be different source code files and subdirectories within the `inc/` and `src/` directories.

- This Makefile should produce object code under a `obj/` directory and mimic the source code's directory structure. For example, `src/sourcelib/coolthing.cpp` would have its corresponding object code file at `obj/sourcelib/coolthing.o`. This Makefile should handle creation of this directory. `mkdir -p` can help out with this (check out the manpage!). There's also a neat Makefile function to get the directory part of a path ;) Note that when you do some pattern matching or substitution reference, that we can change the text e.g. `%.c` and `%.o`: note that we specify `.c` on one side and `.o` on the other. What if you added more non-matching text, such as say, a directory?
- This Makefile should have and use a `CXX` variable set to `g++` for the compiler.
- This Makefile should have and use a `BIN` variable set to `app` for the output executable.
- This Makefile should have target dependencies set up so that individual object code targets can be run and the output executable target can be run to trigger the building of all dependencies.
- This Makefile should have a **phony** `all` target that builds the output executable. This target should have any necessary dependencies.
- This Makefile should have a **phony** `run` target that runs the output executable. This target should have any necessary dependencies i.e. `make run` from a clean state should not error out and should build the output executable before running it.
- This Makefile should have a **phony** `clean` target that deletes the `obj/` directory and the output executable.

When you're done, add and commit your Makefile to the submission repo.

2 Using another build system (3)

In the material we alluded to the existence of other build systems. CMake is another tool that helps generate data needed for other build systems Make. Look up how to work with CMake and create a single `CMakeLists.txt` file that can handle the `example-project`. The requirements will be much more lax for this part: this `CMakeLists.txt` file just needs to build an output application called `app`. (In reality, multiple `CMakeLists.txt` is a common way of handling more complicated directory structures like this.)

- **Make sure not to require an extremely new version of CMake: 3.0 should suffice.** You probably aren't using the latest and greatest features of the latest version of CMake. The course server's (and many build servers' you might encounter) version probably isn't going to be the absolute latest release.
- Like in part 1, the `CMakeLists.txt` is meant to sit inside of `example-project` when run.
- Don't run `cmake .` inside the same directory as the `CMakeLists.txt` file: instead, create a `build` directory, `cd` into it, then run `cmake ..`: this will keep all the build files in a nice, tidy space, and won't overwrite your `Makefile` from the previous section.

Add and commit your `CMakeLists.txt` to the submission repo.

3 Conclusion

1. Commit your Makefile and/or your `CMakeLists.txt` file(s) to your submission repo.
2. Create a file called `report.txt`.
3. On the first line provide an integer time in minutes of how long it took for you to complete this assignment.
4. On the second line and beyond, write down what you learned while doing this assignment. If you already knew how to do all of this, put down "N/A".
5. Add and commit this `report.txt` file. Push your commits to a branch called `build`.
6. As this is a GitLab assignment, remember the autograder is available!