Advanced - Editors

EECS 201 Spring 2026

Submission Instructions

Answer the bolded text that begins with a "**Q**". This assignment is an "online assignment" on Gradescope, where you fill out your answers directly.

Presentation/Checkoff

In addition to submitting this assignment, you will need to attend a lab session after the deadline to discuss your work with the course staff. Refer to the course schedule for the window for doing so for this assignment.

As part of the discussion, be prepared to do the following:

- Explain your approach for the assignment. What documentation did you read, and how did you find it? What did you try that didn't work, and what was your process for figuring out how to fix it?
- Summarize what you learned from the assignment.
- Discuss how you plan on implementing the tools in this assignment and what you learned in your future projects.
- Apply your approach for solving the problems in this assignment to a new problem that you have not seen before.

I suggest keeping these in mind while doing the assignment, and writing down your thoughts so that you can review them before the checkoff.

Your overall score for the assignment will be a combination of your submission score and the presentation/checkoff score. Refer to the syllabus for the details.

1 One Server to bind them (6)

For this part, you may use Vim or Emacs, whichever you are more comfortable with. You must use one of these (or related projects e.g. Neovim): use of a more modern-styled GUI editor like Visual Studio Code or an IDE will not receive credit.

Integrated Development Environments (IDEs), such as Visual Studio, Xcode, or Eclipse or even editors like Visual Studio Code provide many features beyond simple text editing.[1] Some of the most useful features rely on an understanding of C/C++ code, allowing you to jump from a function's use to its definition, inspect the types of complex structures, and assist authoring by suggesting possible completions.

Vim and Emacs both support rudimentary autocompletion out-of-the-box. Try editing an existing file and start typing a function you've used before, then hit `ctrl-n` and `ctrl-p` in Vim or `M-/` in Emacs. This autocompletion has no understanding of C, it simply does string matching with any other strings anywhere else in the open files.[2]

Historically, each individual tool would have to implement its own smart completion tools. Nowadays, as a result of the development of VS Code, the Language Server Protocol (LSP) was born. LSP is a protocol that allows editors to communicate with independent language servers that provide "smart" features for editors, like code completion and refactoring. This has futher empowered and shortened the gap between humble editors and IDEs, the latter of which historically was the implementer of such smart features.

In this assignment, get an LSP set up for your chosen editor going to work with a C or C++ codebase and see how it interacts with a project that you might have (e.g. an old EECS 280 project). This may involve installing a language server to handle C or C++ and installing a plugin for your editor to interact with said server.

1. **Q:** Which editor are you using, what language server are you using, and what did you have to do/install to get the editor working with the language server? If you had to install something, what was it?

[1] Interestingly, "simple text editing" is often what IDEs are worst at out-of-the-box. Hence part 3 of this assignment.

[2] Vim also understands how to parse C `#include` directives and Python `import` statements, and will search those files as well.

2. **Q:** Did you have to do anything to get the language server to understand your project's structure, and if so, what did you have to do?

3. **Q:** For your given editor and its LSP setup, what do you have to do to trigger a feature like autocompletion or syntax error checking? You can describe the necessary keybinds for any intelligent editing feature, not just the two I listed.

2 Extending Editors (2)

For this part, you may use *any* editor you want.
You can also extend your editor with plugins. For Vim, plugins are written in the custom language Vimscript (NeoVim also uses Lua); Emacs itself is a Lisp interpreter, and plugins are written in Emacs Lisp. Sublime is written Python, as are its plugins. Atom in Javascript, et cetera.
While writing a plugin is a great way to learn a lot about the internals of your editor, it can be a steep learning curve. Fortunately, the Internet has an enormous library of really interesting and powerful plugins. As an example, there are plugins that solve the annoyance from part 1, that will automatically regenerate tags whenever needed.
Pick (at least) one plugin(s) to install in your editor.[3] You may find it is easier to first install a plugin manager and then install plugins. **If you installed a plugin for part 1 and 3, you may not count it for this part**.

1. **Q:** What editor are you using, What is this plugin's name, what does this plugin do and why did you install this plugin?

3 Everything Old is New Again (2)

While advanced IDEs and new text editors come out remarkably often, people who have learned how to navigate and manipulate text using Vim or Emacs are generally not willing to give that up. For this reason, there are plugins (or in some cases it's built-in) to provide Vim-like or Emacs-like editing modes for nearly every editor.
Pick your favorite "advanced" GUI editor (suggestions: Visual Studio, Xcode, Eclipse, Sublime Text, Atom, or Visual Studio Code – gVim doesn't count) and set it up to edit text in Vim or Emacs mode. You may need to find and install a plugin first.

1. **Q:** Explain how to set up and use Vim/Emacs mode for your chosen editor

[3] How to find them, choose? Try Googling "best plugins for (editor)", pick whatever seems cool to you.