Project notes

These are some rules and guidelines that apply to all of the projects in this course.

Project domains

Every project handout starts with an overview that sets up the problem. It gives you information about the problem domain that you might not be familiar with, and it sets the goals and motivations for the project. In general, the handout will go out before I've covered all the AI material you'll need to *complete* the project, in part because I will use the ongoing project as a running example in class—which means you'll need to read it as soon as it's assigned, since I'll be talking about it right away.

The Two Biggest Problems

I'm foregrounding this so you are certain to see it. These will be explained in more detail below, but if you submit

- a program that fails to compile, or immediately crashes on all inputs; or
- only once for a final handin with a nontrivial amount of work;

then your **entire project submission grade will be zero**. This fate is easy to avoid! Exercise care!

Checkpoints

After the overview of the project topic, the next portion of each project handout will be checkpoint assignments. The first, which I refer to as "prep work", will typically have little AI content per se but will give you a head start on things like input formats and data structures that you'll need. Which means you can (i.e. should) start on it right away, even before we've finished covering the related AI content in class. If you happen to finish the prep work before its deadline and want to work forward, that's ok but *hand it in* before you start the other work.

The second checkpoint is the "design work", which is to be written on paper (or the equivalent) and brought to class: diagrams, test cases, and other planning work that will help you understand how the program you're writing will need to fit together. I am looking for good-faith best-effort work here, not perfection; we'll talk about everybody's design in class and improve them, so that you'll be well-prepared to really forge ahead on the main body of the work.

Self-analysis: progress reports and "proof"

As you move from the prep and design work into the main body of the project, you should have a plan for how to move from one version that works a little bit to another version that works a little more (rather than trying to type the whole thing in all at once and then debug it). To that end, I'm requiring you to hand in your work regularly, at points where you've completed an identifiable chunk of the work. There's no hard-and-fast rule here, but aim for perhaps 4–6 handins beyond the prep work, after you've done 8–12 rubric points since the last handin.

Each time you hand in (or maybe even more often than that), you should be able to do two things: identify and briefly describe what you've accomplished, and explain what makes you think it works.

To describe what you've accomplished, make sure any AI disclosures are up to date and then add to the bottom of the readme a short note (maybe two or three sentences) with the approximate date/time and a brief summary of what you found tricky about that batch of work and how you problem-solved it (and perhaps note things you're not sure whether they're correct yet), and note any changes you needed to make in the earlier units as a result of your later work. By the final handin you'll have a running log of several of these at the bottom of the readme.

As for what makes you think it works, your "proof"[1] should be organised by rubric line and point to what part of the code, or what test case, or what other evidence you have that shows you've met the criteria for the rubric point. **If your project manager or boss asked why you thought you**

[1]Note the scare quotes—not a proof in the mathematical or CMSC 208 sense

were done, what would you say?

This all does a few things. First of all, this kind of self-reflection seems to make learning "stick" better; second, it's structured in tandem with my AI policies to make it possible to use and experiment with AI assistance but also make it hard to just blindly hand off all the work to the AI.

And in order to make all of this work, I need you to hand in semi-frequently. If you **only hand in once or twice** for work that is substantially beyond the prep work, **you will get zero points** for the progress reports, "proof", and the code itself.

Generative AI

You have the option of not using a large-language-model (LLM) generative AI system on the projects.

However, at least *some* use of generative AI will be permitted on each project, and some of them will (probably) strongly encourage it.[2] If you're at all interested in playing with the LLMs, I generally encourage it—now's the time—but there are a few constraints, laid out below. (Check each project handout to see if there are any adjustments to these rules; this is still very experimental and they may evolve a little.)

- You MUST NOT just paste in the project handout to get a final program to hand in. (That would cause problems with the frequent handin rule anyway.)
- In fact, you SHOULD NOT paste in text from the handout at all (though you can paraphrase it when you're writing your AI prompt)—pasting someone else's work into an LLM is unethical and a form of copyright violation, since you're giving stuff you don't own to the AI company (which they'll then use in training their models).
- Any code that is entirely, largely, or even just partially AI-generated MUST be marked off with comments that indicate what parts were AI generated, which AI (e.g. ChatGPT, Gemini, etc) provided them, and the prompts used, via the "share" link on the chat or edit session. If your AI use is a single long chat window, a single valid share link in

[2]Still working out details on this. Stay tuned.

the readme would be fine, but if your interactions are more targeted you'll want the share link in a code comment in the vicinity of the code that was AI-influenced.

- Remember that you should be working one piece at a time, so your AI requests SHOULD be sort of focused, and talking about how the AI helped you SHOULD be part of your self-analysis in the progress report.

Code submission

The program code you submit can in general be in any language that runs on the lab systems. It's a good idea to make sure the code compiles/runs for each of the intermediate submissions, but it is vitally important that the final submission [compiles and] runs.

If your **code fails to compile** or immediately crashes on all inputs, the **submitted code will receive a zero**.

That means if you test your code and then "just add a quick comment" or some other quick change... you should test it again before handing in.

Documentation

Your project must have documentation. It's nice if it's in a single file called `README.txt` but if it's spread across multiple files, just make sure I can find it all. The things that absolutely must be in the documentation are:

- how to compile/make/setup your program (if relevant)
- how to run your program
- how to run your tests (if relevant)

with real, concrete things that can be typed on the command line (and preferably, with those things on lines by themselves in the documentation, for easy pasting).

Once you've got documentation, though, that's a good place to put anything else you want me to know about. The required documentation, the progress reports, and the "proof" can all go in the same file (but again, if they're in different files, just make sure I can find them).

Project followup

A short time after each project is due, and I've given you at least initial feedback, I will give an in-class followup assessment on the project. The point of the project is to learn some things; and the followup will then assess if you actually did. Is it an exam? Kind of, but with a very practical style and very narrow focus; my intent is that simply *doing the project* is all you should need to do to "study for" the followup. I aim to calibrate it so that if you legitimately work through the project, even with a bit of human or AI help, you'll get roughly the same grade on the followup as on the submitted code (but if someone were to blindly submit code produced by AI or another human, without understanding it, they'd have a hard time on the followup).

Likely question formats include: draw a diagram of a thing from the project; write a test case illustrating a particular element of the project; write a function or method to do something from the project; look at your own code from the project (I'll provide it) and edit it to do a related thing; find and fix the bugs in an implementation of something from the project.

The followups for Projects 0 and 2 will be standalone and take up only a portion of the class day; the followups for Project 1 and 3 will be given together with the midterm and final exam respectively.

This format is still a bit experimental but it mostly worked in 262 last fall. I do appreciate feedback on how you think it's going (and if it goes off the rails I reserve the right to fix it!)

Timeline

As a general rule, projects will be due three weeks after they start. The checkpoint will be due after the first week; they're due **by 8pm** on the due date. (This gives you a chance to ask questions in class and go back and fix things.)

Generally the design work will be due at the start of class the next class period after the prep work. I'll check it in but may not spend a ton of time on it if you don't have specific questions (so make sure to bring clear questions to ask!)

The final handin for a project is due also **by 8pm** on the due date.

Grading

Project grades are out of 150 total possible points.

The **prep assignment** is worth 10 points. You get 10 points if it [compiles,] runs, and it does at least approximately what it's supposed to. You get 5 points if it [compiles,] runs, and does something relevant to the task. You get 0 points if it doesn't compile, if it immediately crashes, or if I can't easily figure out how to make it work. (So, you should include some documentation.) If your prep work gets a zero, work with me ASAP and you can get back up to 5 of the 10 points. (The purpose is to get you started, not to create a penalty!)

The **design work** is worth 10 points, and is graded for completion rather than correctness. If you're absent on the design day and haven't made some sort of advance arrangements, that will normally be a zero on the design work.

The **documentation** is worth 10 points if there's enough documentation that I can *easily* see what to do with your code and what your code does. The more I have to work to figure out what your code even does, the lower this gets. If you have no documentation that I can find, you get zero points for documentation.

The **progress reports** and **"proof"** are each worth 10 points, graded on the same 10/7/4/0 scale as homework problems; submissions that are even slightly beyond the level prep work are eligible for full credit on progress reports and "proof", as long as they're clear and sensible for the project work that was done. (For these points, you're not evaluated on whether you have made *good* design decisions (though I hope you do), but rather on how you explain why you made the choices.)

The **code submission** is worth 40 project-specific points, and the **project followup** is worth 60.

That adds up as follows:

10	prep work
10	design work
10	documentation
10	progress reports
10	“proof”
40	submitted code
60	project followup
150	

The goal will be to calibrate the rubrics so that if you get all of the first 50 points and do roughly as well on the followup as on your submitted code, you’d be getting

- a low D for not making it past the checkpoint
- a middle C for a program that does some relevant thing
- a high B for a program that avoids major pitfalls and correctly solves at least a simple version of the project problem
- a high A or full credit for a program that really gets into the conceptually interesting parts of the problem.

I’ll put rubric specifics in each project handout. Note that I’m not planning to, in general, read your code or comment on it after you hand in—if you want substantive feedback on your code, see me while you’re working on it. I will assign scores based on reading your docs and running your code, but if you’re serious about the self-evaluation in your “proof” file, you should be able to predict your score within a few points based on the rubric I give you in advance.

Each project will be worth 10% of the final grade; the best of the three non-warmup projects will get boosted to count for 20% (i.e. double).

Collaboration and citation

The projects in this course are collaborative, meaning you can (and should!) discuss your ideas with other students, but the code you write needs to be your own. See my collaboration policy for many examples of what is and isn’t acceptable.

The nature of this course and these projects also makes it more reasonable to look on the internet for explanations, algorithms, and even code examples. That's great! But it also makes it more important than ever that you cite your sources. Make sure you read my citation policy for when and how to do this for programs.

Handing in

There is a `handin` command on the lab machines that you will use to hand in all your work. If you use your own computer for development, transfer your files to the lab machines, *verify that they still work*, and then hand them in by typing

```
  handin cmsc389 proj0 file1 file2 file3
```

or

```
  handin cmsc389 proj0 dirname/
```

or

```
  handin cmsc389 proj0 .
```

to hand in an entire directory of files, replacing `proj0` with the actual name of the assignment.

Don't forget to include your documentation, and make sure it's easy for me to find it. `README.txt` is a great name for a file of documentation that you want me to be able to find.

Use the same project name for both the checkpoint and the final handin.