ECE 2400 Computer Systems Programming, Fall 2021
Programming Assignment Logistics

`http://www.csl.cornell.edu/courses/ece2400`
School of Electrical and Computer Engineering
Cornell University

revision: 2021-10-07-12-37

This document describes what students are expected to submit for the programming assignments, and how their submissions will be evaluated at a high level. A handout is provided for each programming assignment that describes the motivation for the assignment and provides background on the required implementations, testing strategy, and evaluation. The handout also describes in more detailed the specific requirements for each PA report. Each programming assignment requires you to submit three parts: the PA milestone (i.e., one or more initial implementations along with the corresponding tests); the PA code (i.e., all implementations along with all tests); and the PA report (i.e., a 4–6 page document with several sections possibly including an introduction, testing strategy discussion, alternative implementation discussion, complexity analysis, quantitative evaluation, optimization discussion, and conclusion). Although each of these parts is graded separately, it is also useful to consider their relative importance to your final grade. The PA code is worth 20% of your final grade; the PA reports are worth 10% of your final grade; and the PA milestones are worth 5% of your final grade. In total, the PAs are worth 35% of your final grade.

Any programmer can write code. *Good programmers* can write code for multiple implementations, use a testing strategy to verify these implementations are correct, and evaluate the performance and space usage of these implementations. *Great programmers* can do all of these things, but can also explain their how their implementations work at a high level, justify the specific design choices used in their implementations, use an evidence-based approach to make a compelling argument that their code is correct, and both qualitatively and quantitatively compare and contrast different implementations. Doing well on the PA milestones and code means you are making progress towards being a good programmer. Doing well on the PA milestones, code, and reports means you are making progress towards being a great programmer. By the end of the semester, we hope every student will have evolved from simply being a programmer to being a *great programmer*.

1. PA Code Release

Initial code for each PA will be released through GitHub, and students will be using GitHub for all development related to the PA. Every student will have his or her own private repository for the first three PAs. Each group of two students will have their own group repository for the last two PAs. All of these repositories will be part of the `cornell-ece2400` GitHub organization, and all development must be done in these specific repositories. **You should never fork your remote repository! If you need to work in isolation then use a branch within your remote repository.** The course instructors will merge new code into the each of these remote repositories, and then students simply need to pull these updates.

2. PA Code Quality

Code quality is one of the criteria used to assess the PAs. Students are required to follow the course coding conventions:

- `https://cornell-ece2400.github.io/ece2400-docs/ece2400-coding-conventions`

Many software companies and open-source software projects use code autoformaters to automate the process of formatting their code according to a set of rules. One such tool is called `clang-format`, and we provide students a `.clang-format` file that adheres to the course coding conventions. To help automate this process we have provided an `autoformat` make target that you can use like this:

```
% cd ${HOME}/ece2400/repo/pa-name
% mkdir -p build
% cd build
% cmake ..
% make autoformat
% git diff
# ... check all changes ...
% git commit -a -m "autoformat"
```

where `repo` is either your NetID for the first three PAs or your group repo for the last two PAs, and `pa-name` is the name of the PA (e.g., `pa1-math` for the first assignment). Note that the `autoformat` target will only work if you have already committed all of your work. This way you can easily use `git diff` to view the changes made by the autoformatting and commit those changes when you are happy with them. Since we provide students an automated way to format their code correctly, students have no excuse for not following the course coding conventions!

Note that students must remove unnecessary comments that are provided by instructors in the code distributed to students. Students must not commit executable binaries or any other unnecessary files. The `autoformat` target will not take care of these issues for you.

3. PA Code Submission

The PA milestone and the final code will be submitted via GitHub. If you are trying to submit your code at the last minute, then it is possible these last minute changes may not make it into your finalized submission. You should make sure your final code is pushed to GitHub well before the deadline.

You should browse the source on GitHub to confirm that the code in the remote repository is indeed the correct version. Make sure all new source files are added, committed, and pushed to GitHub. You should not commit the `build` directory or any generated content (e.g., object files, executable binaries, unit test outputs). Including generated content in your submission will impact the grade for the PA milestone and code quality. Here is how we will be testing your milestone:

```
% mkdir -p ${HOME}/ece2400/submissions
% cd ${HOME}/ece2400/submissions
% rm -rf repo
% git clone git@github.com:cornell-ece2400/repo

% cd ${HOME}/ece2400/submissions/repo/pa-name
% mkdir -p build
% cd build
% cmake ..
% make check-milestone
```

where `repo` is either your NetID for the first three PAs or your group repo for the last two PAs, and `pa-name` is the name of the PA (e.g., `pa1-math` for the first assignment). Some PAs may also require students to pass `memcheck-milestone`.

Here is how we will be testing your final code submission:

```
% mkdir -p ${HOME}/ece2400/submissions
% cd ${HOME}/ece2400/submissions
% rm -rf repo
% git clone git@github.com:cornell-ece2400/repo

% cd ${HOME}/ece2400/submissions/repo/pa-name
% mkdir -p build
% cd build
% cmake ..
% make check
% make eval
# ... run the eval programs ...
```

Some PAs may also require students to pass `memcheck`. If, for any reasons, the above steps do not work, then the score for code functionality will be reduced. For example, students occasionally forget to commit new source files they have created in which case these new files will not be in the remote repository on GitHub.

We will be using GitHub Actions to grade the code functionality for the PAs. So in addition to verifying that a clean clone works on the `ecelinux` machines, you should also verify that all of the tests you expect to pass are passing on GitHub Actions by visiting the GitHub Actions page for your repository:

- `https://github.com/cornell-ece2400/repo/actions`

where `repo` is either your NetID for the first three programming assignments or your group repo for the last two PAs. If your code is failing tests on GitHub Actions, then the score for code functionality will be reduced. Keep in mind that in the final few hours before the deadline, the GitHub Actions work queue can easily fill up. You should always make sure your tests are passing on the `ecelinux` machines and not rely solely on GitHub Actions to verify which tests are passing and failing.

4. PA Test-Case Crowd Sourcing

While a comprehensive test suite provides strong evidence that your implementation has the correct functionality, it is particularly challenging to write high-quality test cases for all of your implementations. Students can use **test-case crowd-sourcing after the milestone** to reduce the workload of constructing a comprehensive test suite. Test-case crowd-sourcing will use a Canvas discussion page; students cannot see any of the currently posted test cases until they post one of their own. Focus on uploading one or two very strong directed or random test cases. Do not upload more than two test cases. Avoid uploading simple directed test cases since students will have already developed such test cases for the milestone. Posting the basic test case provided by the course instructors, posting an obviously too simple test case, and/or posting something which is obviously meant to "game" the system is not allowed. Let's all work together to crowd-source a great test suite that every student can take advantage of!

You can use test cases posted in the Canvas discussion page in your test programs as long as you acknowledge the author, so be sure to include the comment in your source code which describes the test case and includes the author's name. You will need to renumber the test cases and call them correctly from `main()`. **Make sure you understand the test case and that you feel it is testing correct behavior before including it in your test suite!**

5. PA Code Revision

After the deadline for submitting the final code, the course instructors will branch your submission and create a pull request on GitHub. The instructors will then commit the instructor tests and evaluation program into this pull request which will trigger a GitHub Actions build. This will enable the instructors and the students to immediately see how their submission does on both the student tests and the instructor tests. If the student's PA fails some of the instructor tests, then the students are free to fix bugs and commit these changes as part of the pull request. The students are encouraged to add comments to the pull request indicating what they had to change to pass the instructor tests, and why the student tests did not catch this bug. This code revision *will not* mitigate a reduction in the code functionality score due to failing instructor tests, but it *will* enable the course staff to judge how severe a penalty to access. If it turns out that after the students fix a very small mistake in their code, their programming assignment now passes all of the tests then this will result in a small penalty. If it turns out that the students have to fix a major mistake, then this will result in a larger penalty, but at least the students will have figured out what is wrong. Such code revisions will need to be made within a few days of the deadline.

6. PA Report Submission

In addition to the actual code, we also require students to submit a PA report. The report offers an opportunity for students to convey the high-level implementation approach, motivation for specific design decisions, complexity analysis of different implementations, and qualitative evaluation of performance and space usage trade-offs. We would argue that the ability to convey this information via a technical report is just as important, or potentially even more important, than simply writing code.

The PA report should be written assuming the reader is familiar with the lecture material and has read the PA handout. You might need to paraphrase some of the content in the handout in your own words to demonstrate understanding. Details about the actual code should be in the code comments. The report should focus on the high-level implementation and evaluation aspects of the assignment. All reports should include a title and the name(s) and NetID(s) of the student(s) which worked on the assignment at the top of the first page. Do not put this information on a separate title page. The report should be written using a serif font (e.g., Times, Palatino), be single spaced, use margins in the range of 0.5–1 in, use a 10 pt font size, with a page limit of 4–6 pages depending on the PA. All figures must be legible. Avoid scanning hand-written figures and do not use a digital camera to capture a hand-written figure. Do not just use a screen capture of code. Definitely do not include screen captures that have white text on a black background. This is not an appropriate way to include code in a technical document. Cut and paste the code into your report and format it appropriately. Clearly mark each section with a *numbered* section header. Your report should not look like an outline. It should look like a report with paragraphs and prose. Avoid subsections unless there is a very compelling reason to include them.

Each PA handout will describe more details about expectations specific to that PA's report including the specific page limit. Each report will include the specific sections as shown below.

Section	**PA1**	**PA2**	**PA3**	**PA4**	**PA5**
Introduction	✓	✓	✓	✓	✓
Testing Strategy	✓				
Optimizations				✓	
Alternative Implementation					✓
Complexity Analysis		✓	✓	✓	✓
Quantitative Evaluation	✓	✓	✓	✓	✓
Conclusion	✓	✓	✓	✓	✓

General expectations for each section are discussed below, but remember that each PA handout will describe more details about expectations specific to that PA.

- **Introduction (1 paragraph maximum) –** All reports must start with an introduction. Students must summarize the purpose of the PA. Why are we doing this assignment? How does it connect to the lecture material? There are often many purposes. Think critically about how the assignments fits into the other PAs. Students can paraphrase from the handout as necessary. Students must include a sentence or two that describes at a very high-level their implementations. The introduction should be brief but still provide a good summary of the PA.

- **Testing Strategy –** For PA1, students must describe the overall testing strategy (e.g., directed testing, random testing, code coverage). Discuss the difference between white-box and black-box testing. Simply saying the students used unit testing is not sufficient; be specific and explain *why* you used a specific testing strategy (e.g., why use directed testing? why use random testing?). Students must explain at a high-level the kind of directed tests cases they implemented and why they used these test cases. Consider including a table with a test case summary, or some kind of quantitative summary of the number of test cases that are passing. Consider including results from code coverage analysis. Note that students are not required to achieve 100% code coverage. Students are trying to provide a compelling, evidence-based argument that their implementations are functionally correct. Code coverage is just one piece of evidence which should be integrated with the types of evidence (e.g., number of tests, types of tests) in this section. We recommend students start this section with a short paragraph that provides an overview of your *strategy* for testing (so how all of the testing fits together). Then you might have a short paragraph for each kind of testing (one for directed testing and one for random testing). Each paragraph starts with the "why" (why that kind of testing) and then goes on to the "what" (what did you actually test using that kind of testing). Then you can end with a paragraph that pulls it all together with some code coverage data and tries to make a compelling case for why you believe your design is functionally correct. Do not include the actual test code itself; your report should be at a higher level. Do not include the output from running the tests (we can see that on GitHub Actions). **Remember to provide a balanced discussion between how you tested your design *and* why you chose that testing strategy and test cases.**

- **Optimizations –** For PA4, students must discuss all of the optimizations they experimented with. We suggest dedicating a paragraph to each optimization. Start by motivating what overhead the optimization is focusing on, then discuss how the optimization can mitigate this overhead, and finally discuss the details of how the optimization was implemented in your implementations. Students should discuss optimizations even if the optimization did not actually improve performance. Do not quantitatively evaluate your optimizations in this section. **Students must provide a balanced discussion of not just each optimization, but why you chose to apply that optimization.**

- **Alternative Implementation –** For PA5, students must discuss their alternative design in detail. Think critically about what are the key items to mention in order for the reader to understand how the alternative implementation works. Motivate why you decided to pursue this specific alternative implementation. Examples are usually great to include here to illustrate how the alternative implementation works. Students are highly encouraged to include pseudo-code where appropriate. Do not include C code; your report should be at a higher level. **Students must provide a balanced discussion of not just the implementation itself, but why you chose to take this approach.**

- **Complexity Analysis –** For PA2–PA5, students must include a compelling time and/or space complexity analysis of all implementations. Students might need to include a table that succinctly captures the big-O time and/or space complexity of all implementations. While students do not need to explicitly use the six-step method, they must provide a compelling argument justifying their big-O time and/or space complexity results.

- **Quantitative Evaluation –** Students should revisit their complexity analysis from the previous section using quantitative data in this section. Students must report their performance results using a table and/or plot as appropriate. Do not simply include the text output from running the evaluation programs. Format the data so it is appropriate for a report. You must explain how you collected this data (number of subtrials? number of trials? what was the variance?). You must include some kind of analysis of the results: Why is one implementation better or worse than another? For PA4 and PA5 students will likely need to quantitatively evaluate various optimizations. **Remember to provide a balanced discussion between what the results are *and* what those results mean.**

- **Conclusion (1 paragraph maximum) –** All reports end with a conclusion. Students must include a brief qualitative *and* quantitative overview of the evaluation results (Which implementation performed best? By how much? On which inputs?). Students must include some high-level conclusions they can draw from their qualitative and quantitative evaluation. Do not over-generalize. Can you predict how the results might change for other inputs? What can we learn from these results? Which implementation should we use in the future? If it depends, explain why it depends.

It is also always great to include extra material to help demonstrate your understanding. You could include an example of a state diagram like we do in lecture for a small example. You could implement another implementation to gather additional data points to make for a richer comparative analysis in the quantitative evaluation section. You could try more evaluation inputs to illustrate a point. Be sure to highlight "extra" work you did in the introduction. There are many creative things you can do to set your report apart!

Many students initially struggle with the idea of preparing the PA report. In previous courses, students often simply describe their code at a low level in a PA report. In this course, we are challenging students to prepare reports that better demonstrate the student's understanding of the course content. Before starting to write the report, we encourage students to prepare a detailed outline. The outline should include one section for each of the five sections that will eventually make up the report. Under each section, there should be one bullet for each paragraph the student is planning to include in that section. This bullet should describe the topic of the paragraph. Under each bullet there should be several sub-bullets, one for each topic to be discussed in that paragraph. The outline should also explicitly include references to the figures, tables, and plots the student plans to include in the report. This is called a *structured approach* to technical writing. Students are strongly discouraged from "just starting to write". Just like we should always plan our approach before starting to write our programs, we should plan our approach before writing the report. Students are encouraged to review their outline with the course staff several days before the deadline.

Your report should be uploaded to Canavs. We use when the PA report is uploaded to Canvas to track how many slip days students want to use.

7. PA Grading Scheme

The PA milestones are assessed holistically on whether or not both implementations are working and how much effort has been put into the initial testing strategy.

The PA code is assessed using several criteria depending on the PA and weighted as follows:

	PA1	**PA2**	**PA3**	**PA4**	**PA5**
Code Functionality	60%	60%	60%	60%	60%
Verification Quality	30%	30%	30%		
Optimization Quality				30%	30%
Code Quality	10%	10%	10%	10%	10%

As discussed in the syllabus, each criteria/subcriteria is scored on a scale from 0 (nothing) to 4.25 (exceptional work). The functionality of the implementations is assessed based on the number of test cases that pass in both the student and instructor test suites in combination with the severity of any errors. The verification quality is assessed for PA1–3 based on the judgment of the instructor in terms of how well the students' test cases actually test the design. The optimization quality is assessed for PA4–5 based on holistic quantitative analysis of the performance and accuracy of all implementations. The code quality is based on: how well the code follows the course coding guidelines; inclusion of comments that clearly document the structure, interfaces, and implementation; following the naming conventions; decomposing complicated monolithic expressions into smaller sub-expressions to increase readability. Overall, good code quality means little work is necessary to figure out how the code works and how we might improve or maintain the design.

The PA report is assessed using several criteria depending on the PA and weighted as follows:

Section	**PA1**	**PA2**	**PA3**	**PA4**	**PA5**
Introduction	10%	10%	10%	8%	8%
Testing Strategy	35%				
Optimizations				25%	
Alternative Implementation					25%
Complexity Analysis		35%	35%	25%	25%
Quantitative Evaluation	35%	35%	35%	25%	25%
Conclusion	10%	10%	10%	8%	8%
Writing Quality	10%	10%	10%	8%	8%

Again, each criteria/subcriteria is scored on a scale from 0 (nothing) to 4.25 (exceptional work). A detailed rubric will be provided with the PA final report grade to explain how each section was assessed.

8. GitHub and Academic Integrity Violations

Students are explicitly prohibited from sharing their code with anyone that is not within their group or on the course staff. This includes making public forks or duplicating this repository on a different repository hosting service. Students are also explicitly prohibited from manipulating the Git history or changing any of the tags that are created by the course staff. The course staff maintain a copy of all repositories, so we will easily discover if a student manipulates a repository in some inappropriate way. Normal users will never have an issue, but advanced users have been warned.

Sharing code, manipulating the Git history, or changing staff tags will be considered a violation of the Code of Academic Integrity. A primary hearing will be held, and if found guilty, students will face a serious penalty on their grade for this course. More information about the Code of Academic Integrity can be found here:

- `http://theuniversityfaculty.cornell.edu/academic-integrity`