Preparing for the Exam

The CS106B midterm exam is coming up soon. It will be held on Tuesday, February 11th from 7:00PM – 10:00PM, with locations divvied up by last (family) name:

- `A - L`: Go to Cubberley Auditorium
- `M - V`: Go to Bishop Auditorium.
- `W - Z`: Go to 320-105.

The exam will be three hours long. It's closed-book, closed-computer, and limited-note. You can bring a single, double-sided, 8.5” × 11” sheet of notes with you when you take the exam. We've seen all sorts of notes sheets in the past. Some students choose to write down important code patterns and idioms. Others include sample code from their assignments so that they can use them as a reference. We've even seen sheets of paper covered in inspirational messages. Do whatever works best for you – the whole point of the notes sheet is to be useful! We will provide you a reference sheet containing basic syntax of how to use different container types (Handout 10, “Container Syntax Reference”) at the exam itself, so you don’t need to copy that information down.

The topic coverage for the exam is the material from Lectures 00 – 09 (basic C++ up through but not including recursive backtracking) and the material from Assignment 0 through Assignment 3, inclusive. Notice that you are responsible both for the material from lectures and from the assignments, so you should be prepared to answer questions about topics that came up purely on the coding assignments (say, using the debugger, memoization, `structs`, etc.). Topics from later in the course (recursive backtracking, big-O notation, and onward) will not be tested. We will not test you on anything that appears purely in the section handouts or purely in the textbook, though those are excellent resources to draw from when studying.

In terms of the exam format – you should expect the exam to (probably) be four to six questions long, with a mix of short answer questions, multiple-choice questions, and coding questions.

Coding on Exams

During the exam, you will be asked to write code without having the ability to run it. ***This is a fundamentally different experience than writing code in Qt Creator***. Specifically:

- You will not have a compiler that can point out syntax errors.
- You can't run the code, tweak it, and revise it until it works.
- You can't step through your code in a debugger.

This means that the process by which you'll need to problem-solve is different. You'll need to map out where you're going a bit more than what you're used to doing in Qt Creator, and you'll need to be comfortable switching from a high-level idea to something more concrete. Additionally, you'll need to be more detail-oriented than when you're using Qt Creator, since there isn't going to be a compiler to catch your errors for you.

The flip side of this is that we won't be evaluating your code using the standard of "does it compile, run, and work flawlessly?" If we did that, chances are most people would get zero points because a single misplaced brace or stray semicolon would derail everything. Instead, we'll be asking questions like these:

- Do you demonstrate a solid command of the C++ syntax we've covered so far?
- Do the approaches you've taken to the demonstrate a good understanding of the problem-solving strategies we've talked about so far?
- Is your code well-structured in a way that shows a facility with breaking larger problems down into smaller pieces?
- etc.

So, for example, a single missed semicolon or something like that is probably not going to lead to any point deductions, but using the wrong recursive strategy in a recursion problem or the wrong container type in a question on collections would be a more serious concern.

In the context of an exam, we won't be grading for style at the same level as what we'd be looking for on the assignments – we understand that we're essentially grading a first draft. However, you should still aim to make your code easy to read. For example:

- Please use descriptive variable and function names. Single-letter variable names (or worse, single-letter function names) make it significantly harder to understand your code.
- Please properly indent your code. This is especially important in the event that you forget curly braces somewhere and we need to make an educated guess as to what you intended to do.
- If possible, give comments demarcating the different parts of your code. This isn't required, but if you have anything you think is really gnarly, it never hurts to add a comment explaining what you were trying to do!

There are a few details we *don't* care about, so let's save you some time:

- You don't need to add `#include` statements at the top of your code. Assume they're all there.
- You don't need to write function prototypes. You have more important things to worry about. 😃

Preparing for the Exam

There are a number of ways that you can prepare for this upcoming exam. Here is our recommendation of what you should do to get into the best shape that you can.

1. ***Redo the assignments.*** When you're first working on the coding assignments, you're simultaneously trying to figure out what the assignment is asking you to do, solidifying your understanding of the content from lecture, tinkering around to see what happens, and figuring out the necessary problem-solving techniques. That's fine, and that's normal. What matters, though, is that, in that process, you internalized the appropriate techniques and developed your coding and problem-solving skills.

   If you have the time to do so, pick one or two of the most challenging programming questions we asked you to solve, download a fresh copy of the starter files, and *solve the same problem again without referencing your overall solution*. If you're able to do so with a little trial and error, great! It means that you've gotten out of that assignment what we expected you to get out of that assignment. On the other hand, if you're struggling on the assignment a second time, there's a good chance that some key skill or technique hasn't yet clicked for you, and it's worth talking to your SL or stopping by the LaIR to talk through the ideas.

   If you worked with a partner, it's doubly valuable to attempt the problems a second time on your own, just to make sure that you personally are comfortable taking on the questions and that you weren't leaning too much on your partner for insights.

2. ***Work through the section problems.*** The problems we give out in the section handouts each week are a great way to practice specific skills. Want to sharpen your recursion skills? Look at Section Handout 3 or Section Handout 4. Want to shore up your C++ fundamentals? Look at Section Handout 1 and Section Handout 2.

   When you're working through those problems, *don't just hand-write things and call it a day*. You'll want to make sure that you actually got things working. So download a blank set of starter files from the course website, type up your solution, and try running it on some sample test cases. If things work, great! If not, try debugging your code and see if you can fix it. If you're still stuck, no worries! That's a great indicator that you should ping your SL or drop by the LaIR to get some help.

3. ***Review IG feedback and make sure you understand it completely and unambiguously.*** We hold interactive grading sessions on the assignments for a good reason – it's a chance for someone with more coding experience than you (your section leader) to offer their advice and insights about how you can do a better job in the future. If you haven't yet done so, take some time to review the feedback you received. If there are suggestions of the form "try doing it this way next time," take a few minutes and actually go do it the other way. If there are stylistic points that they've pointed out to you, great! Go patch up your code to meet those style guidelines.

   And hey, what should you do if you find something that you don't understand? Ping your SL and ask for some clarification!

4. ***Keep the SLs in the loop.*** As you're studying, please take the initiative to ask us questions when you have them. If you're not sure about how or why a certain piece of code works, or why a certain piece of code *doesn't* work, or why a certain concept works a certain way, etc., go on Piazza or stop by the LaIR and ask us a question. If you worked through any practice problems (section handouts or practice exams), ask your section leader to review your answers and offer polite but honest feedback on how you did and what you need to work on. You can also visit office hours if you'd like!

Exam Policies

We want to be transparent about our grading philosophy for exams and our exam policies. Here's a quick rundown of some of the frequently asked questions about CS106B exams.

- ***Do you give partial credit?*** We do award partial credit on the exams for answers that are on the right track but contain errors. Because you're allowed to bring a notes sheet with you to the exam, we generally do *not* award partial credit for answers that just consist of a lot of code copied from lecture, section handouts, etc. The best way to earn partial credit on the problem is to make a good effort to solve it, and to do so in a way that shows you understand the underlying methodology we've been teaching.
- ***Can I write more than one answer to a problem?*** No, please do not do this. if you start writing out an answer to a problem and realize that it is incorrect, *please cross it off* so that we don't accidentally grade it. If you put down multiple answers to a question, we will grade whichever answer gives you the *fewest* number of points. This policy is in place to prevent "shotgunning" down multiple answers with the hope that one of them will work.
- ***Is pseudocode okay?*** We generally discourage people from writing pseudocode on an exam. It is better to just write real C++ code, or to at least outline in C++ what you would be doing. You should not expect to receive much, if any, partial credit for writing pseudocode.

Good luck on the exam!