Chris Piech
CS 106A

Handout #9
January 2, 2019

Exam Strategies

Based on a handout originally written by Julie Zelinski and Mehran Sahami, with modifications by Brahm Capoor and Julia Daniel

The exams in the CS106 courses can be challenging. Hopefully you have been keeping up in lecture and doing well on the assignments but may be unsure of how to make sure your skills will translate well to the exam setting. The practice midterm gives an idea of what to expect and this handout gives some sage advice gathered from our current and past staff members. We hope you will find our tips useful ☺. Remember: We are assessing your ability to think logically and use appropriate problem-solving techniques. We expect you to express yourself in reasonably correct Java, but we will be quite lenient with errors that are syntactic rather than conceptual.

The Rationale Behind Computer Exams

You will be taking your exam using software called BlueBook. Electronic exams hold several advantages. For students, writing code on a computer (with syntax highlighting, undo, cut, etc) is a much easier exam experience than writing solutions out by hand. The way in which you are assessed is much closer to the way in which you program. The logistics become much easier for students with disability accommodations and students who are taking the course from off campus. On our end, it makes it a lot easier to make, distribute and grade exams. Instead of using the 16,000 pages of paper that CS106A typically consumes each year we use 0. Finally, grading hand written exams is hard and inaccurate. Hand written exams allow for bias in teachers grading [1] and are inaccurate because we can't run your programs. Your midterm will still be assessed by a human, but they will be better equipped to understand what you were trying to do.

Great it's on a computer! Why can't I run my program? In a time-restricted situation, immediate feedback from the compiler can be more of an impediment than an advantage. Imagine this, you read the first problem, have a good idea how to solve it, write your solution, and trace its operation and feel good. You compile and test it. Suppose it exhibits a bug—even though it may only be a minor issue, you can see your answer is wrong—so you hunker down and rework and retest until perfect... even if it takes the whole exam. Bad deal – since you never got to the other problems on the exam! We want you to write your best answer and move on.

How to Prepare For the Exam

"Open book" doesn't mean "don't study". The exam is open-book/open-notes and you can bring along the textbook, Karel reader, your notes from lecture, course handouts, and printouts of all your assignments. We don't expect you to memorize minute details and the exam will not focus on them. However, this doesn't mean you shouldn't prepare. There certainly isn't enough time during the exam to *learn* the material. To do well, you must be experienced at working problems efficiently and accurately without needing to repeatedly refer to your resources.

Practice, practice, practice. A good way to study for the programming problems is to take a problem (lecture or section example, chapter exercise, sample exam problem) and write out your solution under test-like condition. This is much more valuable than a passive review of the problem and its solution where it is too easy to conclude "ah yes, I would have done that" only to find yourself adrift during the real exam when there is no provided solution to guide you!

Get your questions answered. If there is a concept you're a bit fuzzy on, or you'd like to check your answer to a chapter exercise, or you wonder why a solution is written a particular way, get those questions answered before the exam. Swing by the LaIR, come to office hours, or send an email and we're happy to help.

How to take the exam

Scan the entire exam first. Quickly peruse all questions before starting on any one. This allows you to "multitask"—as you are writing the more mundane parts of one answer, your mind can be brainstorming strategies or ideas for another problem in the background. You can also sketch out how to allocate your time between questions in the first pass.

Spend your time wisely. There are only a handful of questions, and each is worth a significant amount. Don't get stuck on any particular problem. There is much opportunity for partial credit, so it's better to make good efforts on all problems than to perfect an answer to one leaving others untouched.

Consider the point value of each question. Divide the total minutes by the total number of points to figure the time per point and use that as guide when allocating your time across the problems. You may want to reserve a little time for checking your work at the end as well.

Leverage the materials you bring with you. If you know of a method/class in the reader or a handout that would help, you can simply cite the source and use it. You do not need to rewrite it. If you have a function you wrote for an assignment that you would like to use, you can copy it from your assignment printouts (hence, why we suggest you bring them).

Style and decomposition are secondary to correctness. Unlike the assignments where we hold you to high standards in all areas, for an exam, the correctness of the answers that dominates the grading. Decomposition and style are thus somewhat de-emphasized. However, good design may make it easier for you to get the functionality correct and require less code, which takes less time and has fewer opportunities for error. Comments are never required unless specifically indicated by a problem. When a solution is incorrect, commenting may help us determine what you were trying to do and award partial credit.

Answer in pseudo-code, but only if you must. If the syntax of Java is somehow in your way, you can answer in pseudo-code for partial credit. There is a wide variation in the scoring for pseudo-code. Some pseudo-code is vague and content-less and does little more than restate the problem description and thus is worth next to nothing. The more details it provides, the better. We typically award at most half of the points for clear pseudo-code precisely describing a correct algorithm. But truthfully, very good pseudo-code contains so much information that it typically would have been easier and more concise to just write in Java in the first place.

Pay attention to specific instructions. A problem statement may include detailed constraints and hints such as "Karel must end up facing East." You may want to underline or highlight these instructions to be sure you don't overlook them. These constraints are not intended to make things difficult; typically we are trying to guide you in the direction of a straightforward and simple solution. If you disregard these instructions, you are likely to lose points, either for not meeting the problem specification and/or for errors introduced when attempting a convoluted alternative.

Syntax is not that important if it is clear what you mean. We won't trouble you about most small syntax errors (forgetting semi-colons, misspellings, etc.) as long as your intentions are clear. Having said that, beware that if your syntax errors cause ambiguity, we might not get the correct meaning. For example, if we see a `for` statement followed by two lines, where both lines are vaguely indented or a third line has been added in after the fact, we may be confused. If there are braces around all the lines, it will be clear you intended both to be a part of the loop body, but without the braces, we can't be sure and it may make your answer incorrect.

Comment out abandoned attempts rather than erasing them. As it usually turns out on a CS exam, you will have false starts on a problem—you try one strategy and hit a dead end. You try something else and then realize you actually were closer to the right solution the first time. If you haven't erased your first attempt, you can always go back to it. Once you work out a better answer, comment out your earlier attempt. When you comment out work, please direct us to the solution you want graded instead. If you appear to have two answers, we will grade which ever is closer to the problem statement.

Save a little time for checking your work. Before handing in your exam, reserve a few minutes to go back over your work. Check for missing initialization/return statements, correct parameters passed to functions, etc. We try not to deduct points for minor things if it is obvious what you meant, but sometimes it is difficult to decipher your true intention. You might save yourself a few lost points by tidying up the details at the end.

Download the exam ahead of time. The exam will be posted Sunday night. Download the encrypted exam before you come to the midterm so that you are ready to go when the test starts.

Prepare your laptop. We will have outlets available, but you may prefer a seat with more space come with a laptop that has enough battery for 2 hours (unless the exam is longer). If you need battery it would be useful for you to bring your own power strip. Make sure your computer isn't just about to run out of memory.

Final Thoughts

Always remember why you are at school. Learning and education tend to be a more fulfilling goal than high grades. If you work hard, study lots and feel good about your understanding of computer science that is an achievement to be proud of—regardless of how many points you get relative to the other students in the class.

Programming is not an easy endeavor, even for experienced programmers. And this midterm may be challenging, but you can do it!