Eric Roberts
CSCI 121

Handout #24
October 1, 2018

Exam Strategies and Tactics

This handout is minimally adapted from one written by my Stanford colleague Julie Zelenski

The exams in an introductory computer science course can be challenging and even a bit intimidating. Although you may have been keeping up in lecture and doing well on the assignments, you may still be unsure of how your skills will translate to the exam setting. The practice midterms give you 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 when preparing for and conquering this upcoming challenge!

The rationale behind pencil and paper exams

Students often suggest that exams should be done more like assignments: using a computer, allowing the use of web-based documentation, being able to run, test, and debug, and so forth. The logistics of an online exam add serious challenges in terms of fairness and security, but we have experimented with online exams in the past. We abandoned those experiments because students did substantially less well on the exams. For a start, much valuable time was lost to dorking with details (proper `import` lines and minor syntax issues) that we don't even count in an exam situation.

In a time-restricted situation, immediate feedback from the compiler can be more of an impediment than an advantage. Imagine yourself in this situation: you read the first problem, have a good idea how to solve it, write your solution, and trace its operation and feel good. In a paper exam, you then move on to the next problem. In an online exam, 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. Writing your first solution took 20 minutes and would have earned, say, 17 of 20 points. You spent another 20 minutes debugging to earn those remaining 3 points. Bad deal! The rest of your exam also suffers because you used up so much time. We have had students who never made it past the first or second problem in an online exam. Being confronted with clear evidence of bugs made it impossible to move on. We tried again and warned students about this effect, but resistance was futile. Given the limited time available, we want you to write your best answer and move on; paper seems to be the means to encourage exactly that.

We know that writing on paper is not the same as working with a compiler, and we account for that in how we design and grade the exam. We are assessing your ability to think logically and use appropriate problem-solving techniques. We expect you to express yourself in reasonably correct Python, but we will be quite lenient with errors that are syntactic rather than conceptual.

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 your notes from lecture, course handouts, and printouts of all your assignments. There will also be printed copies of the reader available for reference.

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 conditions on a blank sheet of paper using a pencil with a short amount of time. 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. Come to office hours or send an email. 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 "multi-task"; 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 while 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 tools you have***. If you know of a function in the reader or a handout that would help, you can simply use it on the exam. 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, which is 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, the correctness of the answers dominates the grading of an exam. Decomposition and style are thus somewhat deemphasized. 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 pseudocode, but only if you must***. If the syntax of Python is somehow in your way, you can answer in pseudocode for partial credit. There is a wide variation

in the scoring for psuedocode. Some pseudocode is vague and content-less and does little more than restate the problem description ("I would find all Pythagorean triples") and thus is worth nothing. The more details your pseudocode provides, the better. We typically award at most half of the points for perfect psuedocode precisely describing a correct algorithm. But truthfully, good pseudocode contains so much information that it would have been easier and more concise to just write in Python in the first place.

- ***Pay attention to specific instructions.*** A problem statement may include detailed constraints and hints such as "you do not have to get the animation to stop" or "you may assume that the string contains at least two characters." 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 introducing errors when attempting a convoluted alternative.
- ***Syntax is not that important if it is clear what you mean.*** We won't trouble you about most syntax as long as your intentions are clear. But if there is ambiguity in your attempt, correct syntax can help us 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.
- ***Write in pencil.*** CS exams done in pen are often messy and difficult to grade. Your first draft may have "typos" (e.g., missing arguments in function call, statements out of order). In pencil, you can easily erase to make the necessary corrections; in pen, it is hard to make such changes and still keep your intentions clear.
- ***Cross out abandoned attempts rather than erase them.*** As is usually the case 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, cross out your earlier attempt. When you cross out work, please direct us to where you have written the solution you want graded instead.
- ***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 (although there are fewer pitfalls in Python than there are in most languages), 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.

Good luck!