Frequently Asked Questions

Q: Why is the Introduction to C Programming class so difficult? Why does it move so fast towards the end of the class?

The equivalent class at MIT, my undergraduate institution, covered three times as much material and enrolls students who have never programmed. The equivalent class at University of Wisconsin-Madison, which I taught as a graduate student, covers twice as much material and is given to beginners. I realize that UCF is simply not in the same league as those schools, but we need to produce competent students that have some chance of competing with better trained students. We CAN'T do that if we completely water down the introductory programming course. It was expected at both of those other schools that students would spend about 10-15 hours per programming assignment. At MIT, students were given 10 in a semester and at Madison, they were given 6. Based on student feedback here, my students spend an average of 4-6 hours per assignment on 6 assignments. Thus, even though COP 3223 seems difficult, it's not difficult in comparison to equivalent courses at other universities. It is, of course, difficult compared to other introductory courses at UCF. Introductory Computer Science courses tend to be more difficult than other introductory courses at other schools as well. Simply put, learning the fundamentals of computer programming requires a fair amount of practice and studying, more than many other disciplines. It's simply the nature of the beast. As for why the end of the class seems to move fast, it's because the new concepts introduced build on previous concepts. If you understand the previous concepts, then the new concepts are very minor and should not take too much time to teach. I fully explain the new concepts in class. The problem is that students have difficulty because they don't understand the previous concepts well and have real trouble when concepts are mixed together. As an example, students frequently ask me, "How do you read information into a struct?" If someone actually understands how scanf works, then there would be no need to ask this question. Once I show you how a struct is defined, and how to access components of a struct, the answer to this question is self-evident based on previous knowledge about scanf. Thus, it appears that the course is moving very fast towards the end, because students have trouble incorporating old concepts with the few new ones being introduced towards the end of class.

Q: Why are experienced programmers required to take COP 3223?

Many beginning students complain about having experienced programmers in COP 3223. Until the university policy changes, the class will continue to have experienced programmers. In no way do these students prevent beginning programmers from performing well in the course. In the past, students who had prior programming experience were allowed to skip COP 3223. Unfortunately, these students frequently complained that the ensuing classes were too hard. Since we had a problem with these students, the policy that made the most sense was to no longer count credit from high school or community colleges in lieu of this class. While it might be nice to revisit this policy, my advice is simply to do the best that you can. Work hard and be sincere. If you are an experienced programmer, don't slack. I've given B's and C's to too many students who claimed to be experienced programmers. This should never happen. On the flip side, I’ve given many A’s to beginning programmers and in many of my classes, the student with the highest grade is a beginning programmer who beat out 80-100 experienced programmers. Yes, this happens and if you care to hear the individual stories, just ask me.

Q: Why do you give difficult/long exams with tricky questions?

Part of the explanation to this question is in my grading philosophy document. Without repeating everything there, the quick answer to that question is that in the work force, computer scientists have to solve problems they've never seen before. That's why I like asking new questions on exams that can be answered using a combination of techniques taught in class. I want to see which students can synthesize information and create solutions to novel problems. In particular, interviews for top companies such as Microsoft and Google actually involve creative brain-twisters that most interviewees have never seen. I would like UCF students to be competitive for jobs at the top companies. If I don't challenge the students here, then they won't be able to compete with students coming from universities where the professors have been giving their students creative questions throughout their undergraduate education.

I make my exams relatively long for a couple reasons. One is that I like to test the breadth of material that has been covered in the class through the exam, if possible. The second reason is that I want students preparing for an exam beforehand so that they have some knowledge on quick recall. I want to reward these students who have prepared as compared to students who have to derive everything from scratch during the exam because they did not prepare. In the working world employees that have quick recall actually complete more tasks than others and are more productive. I want to reward those students in class. In any event, regardless of the length of an exam, all the students have the same amount of time to take it, so the exam is fair on that account no matter what.

Some questions that students perceive as tricky may in fact be testing important concepts. For example, in my C programming course, I will give tracing questions where I put one equal sign where there should be two, or a tracing question with a missing curly braces that still compiles. It's important to learn what the compiler does with these situations and I want to reward the students who have learned this information. The reason it matters is that the better one is at tracing, the better and quicker they will be at debugging programs. I have seen students not find the = vs. == problem for over 45 minutes. The reason is that those students don't know the difference between the two, so when they see the program behaving incorrectly, they don't know what type of expression could cause the behavior they are witnessing. Students who understand this behavior typically find the mistake much more quickly.

Q: Why are assignments sometimes poorly specified?

Sometimes it's because I have forgotten a detail. Other times it's because I am intentionally not specifying certain things. In the working world, vendors will often poorly specify a problem. In spite of this, programmers must still create a product for them. I want students to get some practice attempting to fill in the holes using their common sense for a poorly specified problem. We routinely do this in our everyday lives. When someone doesn't specify something, we make inferences based on context about what they might have wanted. It's important in my opinion to get some practice with this in classes. How will a student deal with this in the working world if they've never practiced it before?

Q: Why is the grading criteria not always posted before an assignment is due and why are some items in the grading criteria strict or nit-picky?

Some educational theorists support always posting comprehensive grading criteria. I disagree with them. I want to produce independent thinkers, not students who are only capable of looking at a checklist of items while completing their homework. If detailed criteria are given, a student will just look through the criteria and implement each feature. I want to reward students who figure out how to solve a problem with a single statement instead of a loop, without letting them know beforehand. I want to reward students who have good practices naturally, even if it’s not on the listed criteria. The truth is, in real life, a boss will ask you to do some set of things, but in many jobs, going above and beyond what is asked is better. (There are some jobs where you need to do exactly what the boss asks, no more no less.) I want to reward students with good practices, so I reserve the right to put points in a grading criteria to recognize the students who came up with an excellent solution instead of an average one.

Q: Why are longer assignments given in some courses instead of more frequent smaller assignments?

Smaller assignments allow students to focus in on a single concept and allow for instant gratification because a program can be quickly completed after the concept is understood. However, most problems in the working world are not in this format at all. Instead, in the working world, most problems are complex and require the synthesis of multiple concepts to solve. This sort of problem solving is more difficult that solving small problems that involve each of the different concepts indivually. This idea is encapsulated in the phrase, "The whole is greater than the sum of the parts." Once again, if students don't get this practice in classes, they will be in for a rude awakening when they get a job. Thus, although they are more difficult to plan, I try to give some larger assignments in courses to give students practice in synthesizing multiple ideas in the solution of a single larger problem.

Q: Why are solutions to sample exams not posted?

I don't want to encourage students to just study the solutions to my past exams, because in my mind, this doesn't constitute learning. It promotes memorization which is the worst way to learn in a computer science class. Some things must be memorized, but by and large, one will do much better in a computer science class and job, if they understand overarching ideas.

I still want students to get a rough idea about the types of questions I have asked in the past. This does not limit the types of questions I can ask in the future, but it does give students a rough guideline of the types of things I think are important. However, I always want to leave room for completely new questions – questions that have not appeared in the lectures or any of the past exams, because that's the only way to truly test conceptual learning.

If a student has worked out a question on a sample exam, often times they can simply run a program to verify whether or not their solution is correct. In other instances, they can come to me to look over their hand-written solution and I will give general comments on it. I will not, however, just say, "oh that's right" or "that's wrong."

Q: Why are pop quizzes given in some classes and scheduled quizzes in others?

I vary assessment from course to course, semester to semester. Pop quizzes encourage attendance, and there is a positive correlation between attendance and course grade. This way, I can encourage students to attend without taking a lot of time just to do attendance. Other times, I simply want to make sure students are keeping up with the course material. The scheduled quizzes accomplish this goal.

Sometimes students are frustrated that a quiz contains material from the previous class because that's too recent, or other times they are frustrated becase a quiz contains material from three classes ago, which is too long ago. Students ought to be learning incrementally and not forgetting things they learned two weeks ago. Asking varied questions like this on a quiz rewards the long-term learners and those who immediately keep up with the information in the course. These are, of course, exactly the students I want to reward.

Q: Why do your web pages only have text and links?

I hate fancy web pages with lots of pictures that take more time to load up. I prefer the information by itself. Also, I don't know HTML very well, nor do I want to learn a whole lot of it. There are lots of other things I am more interested in. I see HTML as something that gives me functionality: it allows me to post information that the public can view. I feel that I have that capability and learning more would mostly be for aesthetic reasons which I don't care about.