CMPSC 16, Conrad, Winter 2015 Syllabus

Print-friendly version at: http://www.cs.ucsb.edu/~pconrad/cs16/15W/pdf/cs16_W15_syllabus.pdf

A Few Course Policies In Brief—more detail later

- If you are registered for another UCSB course that overlaps with this one, you MUST HAVE specific written permission from both instructors, or I am within my rights to give you a failing grade on any work you miss as a result, and will NOT make any accommodations for you. This includes exams.
- You are permitted one sheet of notes on exams—details later on the syllabus.
- Collaboration is only permitted when specifically allowed for—otherwise, you must do your own work.
  - On most homework assignments you may collaborate with at most one other person (who must be named)
- Attendance is required at all lectures and labs (discussion sections) and is checked via homework submission.
  - **you** must hand in homework in class—you may not turn in homework on behalf of another student, or ask someone else to turn in yours
  - If you need to miss class *once*, there is a provision for making up the homework (explained later in this syllabus). The short version: you have to complete the homework, and bring it to the instructor or TAs office hours within one week of when you missed class, in person, and you must stay until it is graded AND RECORDED IN GAUCHOSPACE. And you can only do this ONCE per quarter, except in unusual circumstances.

You may NOT:

- turn in homework on a day other than when it is due
- have someone else turn in your homework for you (that will be considered academic dishonesty).
- leave homework in a mailbox or slide it under a door
- drop it off with the instructor to be graded later.

Grading

40% Assignments/Quizzes/Homework + 40% Midterm Exams (2 at 20% each) + 20% Final Examination

Quizzes may occur at anytime, announced or unannounced. Missed quizzes may not be made up.
Thus **attendance is required**, and **reading the assigned readings is required.**

To determine your final letter grade, I will calculate two numbers:

Weighted course average	40% Assignments Labs/Worksheets/Homework/Projects + 60% Exams (2 midterms and final, 20% each)
Weighted exam average	Average score of two midterms and final (weighted equally)

I will then use the following table. This is a conventional 10 point scale with +, - with the lower three and upper three points of each range representing plus/minus. It also enforces a policy that your final course grade can be no more than one letter grade higher than your exam average.

to earn this letter grade	weighted course average must be at least	AND, weighted exam average must be at least	to earn this letter grade	weighted course average must be at least	AND weighted exam average must be at least
A	93	83	C	73	63
A-	90	80	C-	70	60
B+	87	77	D+	67	57
B	83	73	D	63	53
B-	80	70	D-	60	50
C+	77	67	F	weighted course average below 60 **OR** weighted exam average below 50	

Curving: The grade scale above represents the *minimum* letter grade you will be assigned—at the instructor's discretion, the grading scale may be altered *in the students' favor* if this will be better reflect the students' mastery of the material. Thus, *if* there is a "curve", it will be applied at the *end*, not to individual assignments.

A+ grades: These may be awarded to the very best performing students in the class—but the cutoff for A+ grades will be determined at the end of the course at the discretion of the instructor (there is no pre-determined cutoff---an average of 97 or more doesn't guarantee you an A+ grade.)

What this course is about

This course is the first in a three course sequence, CS16-24-32 that provides a foundation in data structures and algorithms for deeper study of Computer Science.

This is NOT an introductory programming course. This is an INTERMEDIATE programming course.

What you need BEFORE you take this course

This course will present C++ from the beginning; no prior knowledge of C++ is assumed. However, it IS assumed that you already have successfully completed CMPSC 8, or have an equivalent background in programming. You should be comfortable with all of the following:

- Problem solving
  - breaking down a problem into a sequence of steps
  - abstracting specific problems into general ones and finding general solutions
- Memory concepts
  - variables, primitive vs. reference variables, name, type, value
  - assignment statements
  - scope of variables
- Control structures
  - for loops, if/else, while loops
- Arrays (or a similar data structure, e.g. Lists in Python)
  - index vs. value, finding sum, min, max, average, count of elements matching some condition, making a new list of elements containing only those that match some condition
- Functions
  - function call vs. function definition
  - formal vs. actual parameters (arguments)
- Testing
  - How to test your code
- Input/output concepts
  - Writing to the terminal
  - Reading from the keyboard
  - Reading and writing to files
  - Neatly formatting output
- Program style
  - How to write code that other people can read and understand

What you SHOULD HAVE LEARNED BY THE END of this course to be ready for CS24

So, what is it that you need to know by the end of this course? Here's the list of just a few of the things you'll need to know to be ready for CS24 (the next programming course). You'll have the opportunity to learn all of these things (though not necessarily in this order).

- A few of the basic data types of C++, including at least, int, double, char, bool, string
- The basic control structures of C++ (if/else, while, for etc.)
- Defining functions in C++, and passing parameters to functions in three different ways (by value, by pointer, and by reference)
- Scope and lifetime of variables in C++
- The use of "const" with parameters to functions
- Using arrays in C++, and C-strings (null-terminated character arrays)
- How arrays are passed to functions, and the relationship between arrays and pointers
- Defining and working with structs in C++
- Using structs to create singly linked lists where the space for the list nodes is allocated on the heap
- The difference between space allocated on the stack (e.g. local variables) and space allocated on the heap (with the new and delete operators)
- Converting from binary to decimal, octal, and hex, and back again—and how this relates to how C++ programs store various kinds of data in memory.
- The basic principles of recursion, and some idea of when a recursive solution is appropriate.

The swimming/guitar/painting analogy

You cannot learn to swim, play guitar, or paint from a textbook or a lecture. You can only:

- learn to swim by spending many hours in the pool,
- learn to play guitar by spending many hours playing the instrument
- learn to paint by spending many hours putting brush to canvas.

The same is true of programming. Programming is not a series of facts to be memorized—you cannot "cram" for a computer science exam. You must practice, practice, practice.

How this course works

There are five components to this course, each of which has a special job to do:

- **(1) Reading**—Between each class, you'll have reading to do in the textbook. There is too much information you need to learn in this course for you to get all of it in lecture, so the readings are essential. The reading assignments for the next class can be found on the homework assignments handed out during each class period.

- **(2) Homework**—In almost every class, you'll be given a homework assignment that is due in the following class. (The exception is the class period immediately before an exam, where no homework will be given.)

  These are typically pencil/paper type problems, though sometimes you'll need access to a computer to solve them. If you don't have reliable access to a computer at home (or in your dorm), please plan your schedule so that you can spend time in the CSIL computer lab between classes.

  Homework assignments are completed on paper—they may NOT be submitted electronically—and may only be submitted in the class in which they are due. In this way, they also serve as an attendance check. Therefore, you may NOT turn in a homework assignment "on behalf of" an absent classmate, or have someone else turn in your homework for you—doing so in this course is a form of academic dishonesty.

  Because I realize that sometimes each student must miss a class due to unavoidable circumstances, or illness, each student is permitted **one** "personal day/sick day" (one per quarter) where you may make up a missed homework assignment (without penalty) by coming in person during the instructor or TAs office hours. See details elsewhere on this syllabus.
- **(3) Programming Assignments (Labs)**—Programming assignments (also called labs) are given once or twice a week, and are typically started in the Tuesday lab sessions, and finished on your own time outside of lab. These will typically involve pair programming, which is described later in this syllabus.

- **(4) Lectures**—Learning is something that you do as a student, not something that is "done to you" by a teacher. Therefore most of the learning you will do in this course takes place when you are actively involved in doing something challenging, i.e. during the homework assignments, and labs. Most of the information you will need to do those assignments will come from the reading.

  Therefore, you may ask, what is the purpose of the lectures?

  The purpose of the lectures in this course is to guide you through the readings, homeworks, and labs:
  - to provide an overview of how everything fits together
  - to provide hands-on demonstrations of things you'll do on your own later
  - to provide additional information that is not in the textbook
  - to provide additional explanations about things in the text that might not be clear
  - to provide an opportunity to ask questions, and hear answers to questions asked by others.

- **(5) Exams**—There are three exams in this course—two midterms and a final. You may wonder why there are two midterms in a course that is only ten weeks long. The reason is that I've found that in order to succeed at learning the material in this course, many students need two opportunities for feedback on how they are doing before the final exam. Students sometimes are doing better than they think they are, or not as well as think they are, and the exams provide a "reality check" You can find examples of previous quarter's exams at the following links.
  - http://www.cs.ucsb.edu/~pconrad/cs16/14F/exams/

  But BEWARE---offerings of CS16 may vary in terms of their exact pace, and placement of exams. So these exams are a guide to the style of my exams, but NOT necessarily for specific content. In addition, these exams from prior to F14 used the C programming language rather than C++, and went at a much slower pace, so treat them with even more skepticism.
  - http://www.cs.ucsb.edu/~pconrad/cs16/10S/exams/
  - http://www.cs.ucsb.edu/~pconrad/cs16/10W/exams/
  - http://www.cs.ucsb.edu/~pconrad/cs16/09F/exams/

***Everything* is Cumulative—that's just how CS is.**

Finally—note that just as in a math class, everything we do builds on all the work that came before. So, *everything* is *cumulative*—so, you can't afford to miss any classes unless absolutely necessary. Miss two lectures in a 10-week two-lecture per week course, and you've already skipped exactly 10% of the course—it wouldn't be surprising if your performance (i.e. final grade) in the course dipped by a similar amount.

You may find the workload heavy

As a result, the workload in this course may feel heavy. It may even feel unreasonable compared to your other courses.

However, I assure you that it is not unreasonable, given the goal of making you an skilled beginning programmer. Programming is a skill, and the only way to get good at it is lots and lots of practice, which takes lots and lots of time.

The usual "folklore" rule of thumb is 8–12 hours per week for a normal college class. That means you should expect, at a minimum to put in 5–9 hours per week on this course, on top of the 3 hours 20 minutes you spend in lecture and lab each week. To put it another way, I really should be giving you between 2 and 4 hours of homework between every lecture.

I probably won't succeed in giving you that much work to do, but I really should, because that's the only way you'll learn what this course is supposed to teach! So, I'll do my best. If the work load is less than that, I apologize in advance for disappointing you. :-)

Policies

Attendance

This course moves quickly. So attendance is very important. We'll sometimes cover two or even three chapters in a given week. We need to go at that pace, because during the last week of the quarter, you can't really start anything new, because there isn't time to put it into practice with programming assignments. If you don't put it into practice, you aren't very likely to learn it in any way that is going to stick with you, so there isn't much point in just "going through the motions".

As a result, **there will be something you have to turn in at almost every class**. In this way, attendance is taken, and required. These things you have to turn in will be a combination of in-class activities, and homework completed outside of class, but handed in on paper during class.

Missing in-class activities

If you miss a class, you miss the opportunity for the points on that in-class assignment, or homework that was due. Period.

There is no makeup, except for:

- excused absences arranged and agreed to by the instructor **in advance**, for official UCSB activities
- one "sick-day/personal day" per student, per quarter (see below)

To make up an assignment from a "sick-day/personal-day", you must within one week of the absence, or 24 hours before the final exam, which ever is earlier, come to office hours (this cannot be done in lab). You must come in person with the homework already completed, and you should stay until the homework is graded and the grade is recorded in Gauchospace. You may only do this ONCE per quarter.

In rare cases, if there is a documented family emergency, documented extended illness, documented required court appearance, or other situation beyond the students' control (with documentation) the instructor may grant additional make up days entirely at the instructor's discretion—but this is **not** a guarantee or a right.

UCSB Policy on Academic Honesty

> It is expected that students attending the University of California understand and subscribe to the ideal of academic integrity, and are willing to bear individual responsibility for their work. Any work (written or otherwise) submitted to fulfill an academic requirement must represent a student's original work. Any act of academic dishonesty, such as cheating or plagiarism, will subject a person to University disciplinary action. Using or attempting to use materials, information, study aids, or commercial "research" services not authorized by the instructor of the course constitutes cheating. Representing the words, ideas, or concepts of another person without appropriate attribution is plagiarism. Whenever another person's written work is utilized, whether it be a single phrase or longer, quotation marks must be used and sources cited. Paraphrasing another's work, i.e., borrowing the ideas or concepts and putting them into one's "own" words, must also be acknowledged. Although a person's state of mind and intention will be considered in determining the University response to an act of academic dishonesty, this in no way lessens the responsibility of the student.
> *(Section A.2 from: http://www.sa.ucsb.edu/Regulations/student_conduct.aspx, Student Conduct, General Standards of Conduct)*

About pair programming

Most of the programming work in this course will be done using a style of programming known as "pair programming". This is where two people (in rare cases, three) work together at the same terminal to solve a programming problem.

It is similar, in some ways, to having a "lab partner" in a Biology, Chemistry or Physics course.

For the assignments where pair programming is used, it is required, not optional. Here's why:

- Pair programming is a real-world skill that is highly valued by employers.
  - Many companies use pair programming extensively, including several local area employers of UCSB CS graduates.
- Companies that employ UCSB CS and CE grads tell us that our graduates have good technical skills but need better skills and working in pairs and groups to solve problems.
  - Incorporating pair programming into our curriculum is part of our response to this "real-world" feedback.
- Most students find it helpful and enjoyable—UCSB CS students from 2009-2010 that were surveyed about their pair programming experiences overwhelmingly reported positive results.
- There is also evidence in the scientific literature that it improves student learning, and helps you get better grades.

To learn more about pair programming, watch the following video (it takes less than 10 minutes).
http://bit.ly/pair-programming-video

About Collaboration

<p>As mentioned above, one of the things we really want to convey in this course is that real-world software development is very seldom an 'individual sport'—it is much more often a 'team sport'. Companies want to hire CS and CE graduates that know how to collaborate with others on producing software.

In the CS Department at UCSB, we understand the value of this. However, it puts us in a tricky position.

On the one hand, we want to encourage working together in ways that help you develop your skills and teamwork, and help you understand that programming can be a social, collaborative, creative activity—not something done only by loner nerds in cubicles. The sooner you start with activities such as pair programming, code reviews, and other collaborative software development activities, the more skill you'll develop, and the sooner you'll be ready for the real world. Plus, for many people, working together with others is a lot more enjoyable and fun than being a loner.

On the other hand, we need to avoid any situations where freeloaders are "coasting" through courses by leaning too much on others—never developing independent skills as programmers. This situation creates huge problems. Mostly it is damaging to the freeloaders themselves, who eventually crash and burn, perhaps far too late to choose another career path without significant difficulty. However, it also creates problems for everyone else—some hardworking students become demoralized by the unfairness of it all, and the value of a UCSB education is diminished by the freeloaders' lack of accomplishment.

Thus, we must strike a balance.

Our emphasis on collaboration means:

- We will try to create opportunities for you to work in pairs on assignments—in some cases, we may even require it.
- We will try to create opportunities for you to develop the ability to think critically about software development by talking about and reflecting on your own code and other people's code in small groups (code reviews).
- Some in-class assignments will permit discussion with other students.

However:

- You may not "just copy" homework or code from others and claim it as your own work.
- You may not work together on assignments unless you've been specifically told that it is allowed.

The bottom line:

- We'll try to be very specific about what kinds of collaboration are permitted, and what kinds of collaboration are not permitted, and are considered a form of academic dishonesty.
- If you are not sure about whether some kind of collaboration is permitted or not, it is your responsibility to **ask questions.**

A final note: the emphasis on collaboration in this course does not necessarily extend to other CS courses you may take in the future.

- Each course will have its own policies, and the default policy is still: **no collaboration.**
- Please be sure you understand each instructor's policy on collaboration carefully, and don't assume it will be the same as that from previous courses.
- And, finally, be sure to review the UCSB Academic Honesty Policy. You should read and understand the UCSB policy on academic honesty listed below. You should also understand that I take academic honesty and personal integrity very seriously, and will do my best to uphold and enforce this UCSB policy.

Late Lab Policy

The policy is simple, and is based on the idea that the primary purpose of the deadlines is to allow the TA manage his/her workload. The number of labs in this course requires that he/she not have to do "context switching" between grading different labs. All labs must be graded in one sitting, or he/she just won't be able to keep up with the workload. So:

- If you want your work to be graded without penalty, turn it in on time.
- If you turn in your lab late, you RISK GETTING A ZERO.
- We will grade late labs ONLY if it creates no extra inconvenience for the graders, and we WILL impose a penalty between 10-20% (see the individual grading rubrics for the labs.)
- There is NO GUARANTEE that late labs will be graded at all. The TA will simply start work at some point after the deadline, and grade until he/she is finished. At that time, he/she will "close the books" on that particular lab, and any work not submitted at that time will NOT be considered.

Other frequently asked questions

Do we need to bring our textbook to lecture? To lab?

It is generally not necessary to bring the textbook to *lectures* for CS16. It may be helpful to bring the book to lab though, as it may be a useful reference for working on programming assignments.

Though I say it is not necessary to bring your textbook to lecture, here are two specific cases, though, where it might be helpful:

- If you want to ask a question about something specific from the reading, and you would like to have the textbook to refer to.
- If you happen to have some "unscheduled time" immediately after the lecture ends at 3:15, and you can take that time to do the homework for the next day's class. You'll get the assignment in class, and if you have your textbook with you, then right after class you can start on it right away and get it done. This is the best time to work on it, while the topic of CS16 is fresh in your mind---and starting early gives you more chance to ask me and the TAs questions in case you are not sure about something.
- I offer extra credit to the first person to report typos in the assigned readings from the book, or the homework assignments themselves. Thus, starting earlier gives you a better chance to be the first one to report the typo. There is a special forum on Gauchospace for reporting typos—the time it is first reported there is what counts for extra credit (reporting it by email or in person or in class is *nice*, but does *not* earn the extra credit.)

Do we need to bring laptops to class?

You do not *need* to bring a laptop to class.

Are we permitted to use laptops during lecture?

Some instructors are bothered by students having laptops in class—and I can definitely understand this, especially in cases where students are not actually using the laptops to take notes, or try out code, but instead to read email or Facebook, play games, etc. That latter use can be a real distraction to others sitting behind that student, not to mention the distraction to the laptop user herself/himself.

I recognize the value that having a laptop in class can offer to the student that is using it for legitimate purposes related to the class, so I'm not going to start the quarter with a no-laptops policy. However, if I find that this isn't working out, I reserve the right to change the policy and implement a ban on laptops in lecture (where the only exception would be someone with a medically certified disability that needs the laptop for ADA-compliance.)

So, you don't want to be "that guy" or "that gal" that spoils it for everyone else. If you use a laptop, use it for legit purposes during class time.

What about laptops in lab?

Sometimes students want to use their laptops in lab instead of using the Linux computers provided. I would like to strongly discourage this for a variety of reasons:

- learning to use the Linux environment is one of the learning objectives of the course—that environment is used in later CS courses (CS24,CS32,CS48,CS56, and almost every upper division programming course.)
- using the Linux environment helps facilitate the whole "pair programming" thing—if it's "your" computer, your pair partner is likely to feel less comfortable than you. Using the shared machines "levels the playing field".

This isn't an absolute prohibition. If the Linux machines are having software or hardware problems, or there is a problem with your account—these things happen rarely, but they do happen sometimes—then, sure, go ahead and use the laptop during lab. But otherwise, please stick with the machines provided.

Do we need to download anything to our laptops or computers at home/in the dorm?

If you are using Windows, you definitely will want to download two programs: PuTTY and XMing. These programs allow you to access the Linux environment on the CoE computers from your Windows machine. (Mac and Linux users usually have tools to do that already installed as part of the operating system called "ssh" and "X11".)

Disclaimer

This syllabus is as accurate as possible, but is subject to change at the instructor's discretion, within the bounds of UC policy.