

**Course:** CS 2110 —also ENGRD 2210 (engineers should sign up for ENGRD 2210)

**Course website:**

<http://www.cs.cornell.edu/courses/CS2110/2016fa/>

**Instructors:** Mike George and David Gries

**CS 2111:** A 1-credit S/U course designed to help students who want more contact hours. Only for students in CS2110. It gives more explanation of core ideas in Java, programming, data structures, assignments, etc., and provides more opportunities to ask questions. Requirement: Attend one 1-hour session each week. We recommend it!

**CS 2112:** An honors version of this course, given at the same time. If you know Java and like CS, consider taking it. You can switch back to 2110 after 2–2.5 weeks if you want. (Looks like the course has reached its capacity)

**Academic Excellence Workshop (AEW):** A 1-credit S/U course for students in CS2110 in which students work together in a cooperative 2-hour session each week. Requirement: Attend weekly 2-hour session. For info, visit Olin 167 or this website:

[www.engineering.cornell.edu/academics/undergraduate/curriculum/courses/workshops/index.cfm](http://www.engineering.cornell.edu/academics/undergraduate/curriculum/courses/workshops/index.cfm).

**CS 2110 Description.** Intermediate programming in a high-level language and introduction to computer science. Topics include program structure and organization, object-oriented programming (classes, objects, types, sub-typing), graphical user interfaces, algorithm analysis (asymptotic complexity, big “O” notation), recursion, data structures (lists, trees, stacks, queues, heaps, search trees, hash tables, graphs), graph algorithms. Java is the principal programming language.

**Course outcomes**

- 1: Be fluent in the use of recursion and object-oriented programming concepts (e.g. classes, objects, inheritance, and interfaces).
- 2: Be able to design and implement parts of nontrivial Java programs (roughly 1000 lines of code), starting from an English language specification.
- 3: Understand graphical user interfaces.
- 4: Understand asymptotic complexity of algorithms and be able to analyze programs to determine their running times.
- 5: Understand basic data structures taught in the course and be able to implement them and use them in programs.

**Prerequisite:** CS 1110 or CS 1112 or an equivalent course that provides a solid introduction to a procedural

programming language —see the back of this page. *Knowledge of Java or object-oriented programming is not a prerequisite.*

**Lectures:** Tues/Thurs 10:10–11:10, Statler Auditorium. Attendance required.

**Sections:** Sign up for one 1-hour section. Attend every week. Sections teach new material, review material, give help on assignments, etc. No permission needed to switch sections, but register for the one you attend regularly. Some sections given at the same time are unbalanced with regard to enrollment. Please move to a different section to balance the load, if you can.

**Coursework and grades:** Visit the course page <http://www.cs.cornell.edu/Courses/cs2110/2016fa/courseinfo.html> for an extensive discussion of grades in this course.

**Programming Assignments:** Come in two flavors:

- Vanilla: specific experience to learn and practice what’s being taught. We tell you exactly what to do.
- Chocolate: more open-ended project, in which you get to do design, etc.

For the chocolate assignments, we will leave varying parts of the design to you —it’s more fun but more challenging because you get to make design decisions.

CS 2111 (see left column) will offer extra help and guidance on assignments.

**Course CMS:** The system used to submit assignments, record grades, etc. URL is on course website. Don’t look at it now —wait until we tell you that it has been populated with students.

**Piazza:** The website we use to communicate questions, answers, etc. We hope you will use it a lot, both asking questions and answering questions that other students ask. <https://piazza.com/cornell/fall2016/cs2110/home>

**Exams:** The two prelims and the final are given at the times shown below. Put them on your calendar *now*. Makeups are generally not allowed except in really exceptional circumstances. If you are out of town, arrangements may be made for you to take it while out of town. For more information, visit the course website, click on *Course Info*, and click on *Policies*.

Prelim 1: Thur, 29 Sep. 7:30-9:30PM or 5:30-7:00PM  
 Prelim 2: Thur, 17 Nov. 7:30-9:30PM or 5:30-7:00PM  
 Final: TBA

**Textbook:** “Data Structures and Abstractions with Java”, Prentice Hall, any edition. Copies on reserve in Engineering Library. You may not need your own copy; share with others; it is optional.

**Lecture slides:** We generally make the slides available on the course website (click *Lecture Notes*) the day before a lecture. Please download them the day before the lecture, scan through them to get an idea what the lecture is about, and bring a copy to class (paper, on laptop, on tablet, whatever).

**VideoNote:** Videos of lectures from Fall 2015 will be available on [www.videonote.com/cornell](http://www.videonote.com/cornell)

**Java-OO-Summary.** Extensive ppt slides (75-80 of them), which provide a summary of OO concepts in Java. Has a 2-page index at the beginning. Available from the course website on the Resources page.

**Java:** You need Java 8 (also called 1.8) on your computer. If necessary, download and install it using instructions on the “Resources” page of the course website.

**Eclipse:** A free IDE (Integrated Development Environment) that we use to write, debug, and run programs. The “Resources” page on the course website has instructions for downloading and installing it.

**DrJava:** A free and extremely basic IDE, which we sometimes use to demo things easily. Use it to try various things, but do not use it for assignments. Download from <http://www.drjava.org>. Download a “jar file” version, not the app.

**codingbat.com.** A website where you can practice writing small Java segment and see results immediately. Practice Boolean expressions, strings, arrays, recursion, etc. See the course website on Resources page.

**Academic Integrity:** On any programming assignment, it is a violation of academic integrity to:

1. Look at or be in possession of the code of another group, in this semester or a previous one with a similar assignment.
2. Show or give your code to another student.
3. Post code on the Piazza.

Naturally, you may discuss assignments with others, but the discussion should not extend to writing actual code, picking variable names, agreeing on specifications or comments, etc.

If you do an assignment with another person (in a Group), you must both sit at the computer together, working together. It is a violation of the code to split the work into parts, do the parts independently, and then merge.

### Programming Proficiency

A CS 2110 student should be able to write simple programs in some programming language. They should have fluency with strings, arrays, two-dimensional arrays, simple conditional tests, loops, and functions and

procedures as well as calls on them.

Here are examples of the sorts of programs you should be able to easily write. To test yourself, consider launching the editor you used when you learned to program and actually write, test, and debug them.

1. Write a function that returns true if its string parameter is a palindrome (and false otherwise). A palindrome is a string that reads the same backward or forward, e.g. "Madam, I'm Adam." Actually, this string would fail the test because it contains white space and punctuation. With parameter "madamimadam", the function would return true.

2. Write a function that returns its string parameter but with punctuation and spaces removed and letters turned into lower case. If you call your function from problem 1 with the output of this new function, "Madam, I'm Adam." would pass the test.

Ideally, use some existing string function in the language you are familiar with to test for white space and punctuation and to map upper case to lower. No need to reinvent the wheel.

In CS 2110 we prefer to use the provided language features, including prebuilt library methods, to full effect. The best programmers are the ones who are most effective in using the tools available to them: they write less code, and their code is more expressive and more exact, so they make fewer mistakes.

3. Compute the median of a one-dimensional array  $x$  containing integers, or count the number of zeros in  $x$  (each of these actions would be a separate method, returning an integer value). Compute the mean as a floating point number.

4. Given integers  $b$  and  $c$ , where  $0 < c$ , compute  $b/c$  as an integer (rounded to the nearest integer).

5. Count the number of zeroes in a rectangular matrix  $y$ . For a square array square, determine whether all the diagonal elements have the same value.

6. Define the "balance" of a rectangular matrix  $y$  to be the number of elements larger than the mean value (rounded to an integer using the method of question 4) minus the number of elements smaller than the mean. Given an integer matrix, compute its mean and balance.

7. (Binary search). Given a sorted integer array segment  $b[h..k]$  and an integer  $x$ , find the position  $j$  such that  $b[h..j-1] \leq x$  and  $b[j..k] > x$ . (by  $b[h..j-1] \leq x$ , we mean that all values of  $b[h..j-1]$  are  $\leq x$ ). Your program should run in time proportional to the logarithm of  $k+1-h$ . (Did you have binary search in your previous course? If so, this should be easy. If not, don't worry; we will teach it to you.)